

1. [Tkc0100: Preface](#)
2. Scratch v2.0
 1. [Scr0300: Scratch 2.0 Overview](#)
 2. [Scr0310: Getting Started with Scratch 2.0](#)
 3. [Scr0320: Memory, Variables, and Literals in Scratch 2.0](#)
 4. [Scr0330: Sequence, Selection, and Loop in Scratch 2.0.](#)
 5. [Scr0340: Arithmetic Operators in Scratch 2.0](#)
 6. [Scr0350: Relational Operators in Scratch 2.0](#)
 7. [Scr0360: Logical Operators in Scratch 2.0](#)
 8. [Scr0370: The repeat loop in Scratch 2.0](#)
 9. [Scr0380: The forever and repeat until loops in Scratch 2.0](#)
 10. [Scr0390: Custom blocks in Scratch 2.0](#)
 11. [Scr0400: Highlights from Scratch 2.0](#)
3. Scratch v1.4
 1. [Scr0100: Scratch Overview](#)
 2. [Scr0110: Getting Started](#)
 3. [Scr0120: Memory, Variables, and Literals](#)
 4. [Scr0130: Sequence, Selection, and Loop](#)
 5. [Scr0140: Arithmetic Operators](#)
 6. [Scr0150: Relational Operators](#)

Tkc0100: Preface

Preface to a collection of modules designed to help beginners of all ages learn how to create the code for computer programs. Information is provided not only for the beginners themselves but also for their parents and teachers where appropriate.

Table of Contents

- [Preface](#)
- [Background information](#)
- [Discussion](#)
 - [Many paths are available](#)
 - [The recommended path](#)
 - [Programming knowledge requirements](#)
 - [Won't cover the first three items](#)
 - [Text based vs. drag-and-drop vs. popup programming](#)
 - [Text-based programming](#)
 - [Drag-and-drop programming](#)
 - [Popup programming](#)
 - [Scratch](#)
 - [Links to useful Scratch material](#)
 - [New features in Scratch 2.0](#)
 - [My publications for Scratch v2.0](#)
 - [Are you too old for scratch?](#)
 - [Additional drag-and-drop programming resources](#)
 - [Text based learning resources](#)
 - [Codecademy JavaScript](#)
 - [Codecademy Web Fundamentals](#)
 - [Codecademy Ruby](#)
 - [Codecademy PHP and Python](#)

- [Udacity - Introduction to Computer Science](#)
- [Object-Oriented Programming.\(OOP\) with Java](#)
- [Other options](#)
- [Miscellaneous](#)
- [Appendix A: Programming knowledge requirements in Texas](#)
- [A list of requirements](#)

Preface

This module is one in a collection of modules designed to help beginners of all ages (*8 and up*) learn how to create the code for computer programs. Information is provided not only for the beginners themselves but also for parents, teachers, and other helpers where appropriate.

The purpose of this module is simply to introduce the collection.

Background information

In the year 2013, everything from parking meters to cell phones to Mars landing vehicles are controlled by computer code. That code is written by humans.

[Bill Gates](#) , [Mark Zuckerberg](#) , [Mitch Resnick](#) , and many other [respected individuals](#) are telling us that *"Everybody in this country should learn how to program a computer... because it teaches you how to think."*

If you search the web, you will find a maze of web pages containing information for teaching beginners how to create program code. This collection will attempt to pull much of that information together and to provide explanations that will make it easier for beginners to navigate that maze and to get involved in computer programming.

As mentioned earlier, the information in this collection is intended not only for the beginners themselves, but also for the parents, teachers, and other

helpers of those beginners where appropriate.

Discussion

Many paths are available

There are many paths that a beginner can take and many tools that a beginner can use in learning to write computer code -- some good and some not so good. Here is a non-exhaustive list of tools and resources (*in no particular order*) that I have collected **from the web** :

1. [Scratch](#)
2. [Codecademy](#)
3. [Udacity - Introduction to Computer Science](#)
4. [Alice](#)
5. [CodeHS](#)
6. [Kodu](#)
7. [Python](#)
8. [Pygame](#)
9. [CS1graphics.org](#)
10. [Sugar on a Stick](#)
11. [Lua](#)
12. [OpenSim](#)
13. [Spritely](#)
14. [Edubuntu](#)
15. [App Inventor](#)
16. [Lego Mindstorms](#)
17. [Greenfoot](#)
18. [BlueJ](#)
19. [Java](#)
20. [Play-i](#)
21. [Marshall Brain: Teaching your kids how to write computer programs](#)
22. [Code.org](#)
23. [Treehouse](#)
24. [W3schools.com](#)
25. [TurtleAcademy](#)
26. [StackOverflow](#)

27. [Programr](#)
28. [Mozilla Thimble](#)
29. [MIT Open Courseware](#)
30. [LearnStreet](#)
31. [LCodeTHW](#)
32. [KidsRuby](#)
33. [JSDares](#)
34. [HTML5 Rocks](#)
35. [Hackety Hack](#)
36. [Tynker](#)
37. [Stencyl](#)
38. [Microsoft Small Basic](#)
39. [Snap! \(Build Your Own Blocks\)](#)
40. [Blockly](#)
41. [Kodable](#)
42. [Hopscotch](#)
43. [Object-Oriented Programming.\(OOP\) with Java](#)

The many available options can lead to confusion, particularly for beginners, parents, and teachers without a strong technical background. Different items in the above list are best suited to students of different ages with different backgrounds.

In this collection, I will attempt to provide the information needed to help beginners, parents, and teachers alike sort through the options in order to navigate a path that makes sense for each particular beginner.

The recommended path

Although my recommendation may change over time as new tools and resources become available, as of May 2013, my recommended path is as follows:

1. Begin with [Scratch 2.0](#). You can read more about Scratch and some of the other items on my recommended path in the sections that follow. By the time that you finish with Scratch, you should be able to create

- programs to solve the Blockly [maze puzzles](#). They will test your knowledge of certain aspects of programming logic.
2. When you are comfortable with Scratch, complete the following free online course: [Codecademy JavaScript](#). If you are really ambitious, you might also want to complete [Codecademy Ruby](#).
 3. Once you complete the [Codecademy JavaScript](#) course, take a temporary departure from programming and learn about another form of coding by completing the following free online course: [Codecademy Web Fundamentals](#).
 4. At this point, you will need to graduate into the use of a compiled type-sensitive programming language. My recommendation is to complete all of the *Programming Fundamentals* modules at [Object-Oriented Programming \(OOP\) with Java](#).
 5. Next you should take a look at the *Objects First* module at [Object-Oriented Programming \(OOP\) with Java](#).
 6. Following that, you should study the modules in *ITSE 2321*, *ITSE 2317*, *GAME 2302*, and *Anatomy of a Game Engine* at [Object-Oriented Programming \(OOP\) with Java](#).

There are many other resources in the [above list](#) that are worth pursuing. However, once you reach the end of this [recommended path](#), you will be sufficiently well informed that you will no longer need my recommendations to help you chart your path forward.

Programming knowledge requirements

Several years ago, I extracted information from various Texas State documents in an attempt to get a handle on the minimum level of programming knowledge that is required for students to graduate from a Texas high school with one or more courses in computer science. By organizing that material, I came up with the list in [Appendix A](#) as the minimum list of items that must be understood by the graduating student.

Won't cover the first three items

I probably won't attempt to cover the first three items in the list in [Appendix A](#). Instead, I will recommend that you go on the web or go to a used bookstore to purchase and then study the first few chapters of any one of hundreds of textbooks on programming fundamentals that have been published in the past twenty years that cover those items. That material hasn't changed much in twenty years so it doesn't need to be a new textbook. Furthermore, that material is generally independent of the programming language being used, so it doesn't even need to be a textbook for a specific programming language.

I do plan to cover many of the remaining items in the list in [Appendix A](#) one or more times in this collection of modules but not necessarily in the order given.

This is a work in process, covering a wealth of knowledge, which I will be updating in upcoming months and possibly even years.

Text based vs. drag-and-drop vs. popup programming

Text-based programming

Historically, computer programs have been written by typing text on a computer keyboard, and that remains true even today for mainstream programming. That is what I mean when I refer to text-based programming.

However, in recent years, two alternative ways to create computer code have become popular:

- Drag-and-drop programming
- Popup programming

Drag-and-drop programming

As of May 2013, this form of programming is primarily used in environments that are designed for teaching programming concepts. Drag-and-drop programming hasn't yet made its way into mainstream programming. However, that may be about to change (see [MIT App Inventor](#) and Google [Blockly](#) for example) .

The two primary players in the educational drag-and-drop programming field are [Scratch](#) and [Alice](#) , but they are not alone. Several other players are also on the scene (see [Snap!](#) and [Tynker](#) for example) .

The primary advantage of drag-and-drop programming is that it eliminates the requirement to memorize complex programming syntax and allows the programmer to concentrate on the concepts. Most of the work is done by dragging visual images of blocks (*containing program code*) from a tool box and assembling them into a program. Very little keyboard work is required, which in turn greatly reduces the likelihood of spelling and syntax errors.

Given the current popularity of touch screens and the disdain for keyboards, I see a bright future for drag-and-drop programming within the next ten years, even for large scale commercial applications. [MIT App Inventor](#) and [Blockly](#) are leading the charge in that regard.

Popup programming

Popup programming is a supplement to text-based programming and is not intended for educational use. Instead, it is intended to make mainstream programmers more productive by reducing the amount of typing required.

Many different forms exist, but they generally take on the form of a popup menu that appears at certain critical times to allow the programmer to select code from a menu as an alternative to typing the code.

One form of popup programming is *code completion* . With code completion, the Integrated Development Environment (*IDE*) examines the first few characters typed by the programmer and makes an educated guess

as to the remaining required characters, which the programmer can either accept or reject.

Scratch

I suspect that the all time most successful self-study programming environment for beginners is [Scratch](#). This is probably the result of three factors:

- [Scratch](#) makes it easy to get started programming
- [Scratch](#) is engaging for today's fast-paced multimedia generation
- [Scratch](#) provides a monitored online social network that allows interaction among beginners with a common interest.

In my opinion, the most difficult aspect of teaching coding to beginners is getting their attention and getting them engaged. Therefore, the first group of instructional modules in this collection will be designed to help beginners, their parents, and their teachers get started programming with [Scratch](#).

Links to useful Scratch material

Up until May 9, 2013, the latest version of [Scratch](#) was version 1.4. For several days prior to May 9, the [Scratch](#) website was shut down to allow the folks at MIT to migrate from version 1.4 to version 2.0.

On May 9, 2013, [Scratch](#) version 2.0 was officially released and the [Scratch](#) website was reopened reflecting that change. I will have more to say about the new version of [Scratch](#) later. In the meantime, some of the following links may take you to information about the old version, some may take you to information about the new version, and unfortunately, some may have become broken during the transition.

- Click [here](#) to watch a short video about [Scratch](#).

- Click [here](#) to see what Mitch Resnick, director of the Lifelong Kindergarten group at MIT Media Lab has to say about the need for beginners to learn to code.
- Click [here](#) to view the online version of an interesting [Scratch](#) project named Day Dream.
- Click [here](#) to take an online tour of [Scratch](#).
- Click [here](#) to view online [Scratch](#) tutorial videos from MIT.
- Click [here](#) to view thousands of online scratch projects.
- Click [here](#) to learn more about scratch.

New features in Scratch 2.0

There are a number of new features in v2.0, some of which are very nice. I will refer you to this [preview article](#) that discusses some of those new features.

Internet access is required

There are a couple of critical aspects of v2.0 that I will mention. Although there was a strong web component that involved sharing projects with Scratch v1.4, that version of [Scratch](#) was a program that was downloaded, installed on the local machine, and run locally. Once the program was installed on the local machine, Internet access was nice but no longer required.

Scratch v2.0 requires Internet access to be of any use. The development program runs online in a browser. Therefore, those without Internet access and those with poor Internet access won't be able to use Scratch v2.0.

Version 1.4 is still available

Fortunately, as of May 2013, it is still possible to download and install v1.4 from http://info.scratch.mit.edu/Scratch_1.4_Download. Then the program can be run locally without a requirement for Internet access.

The Adobe Flash player is required

Another critical aspect v2.0 is that the Adobe Flash player must be installed on the local machine to use v2.0. While this may not be a problem in most cases, it is something that potential users of v2.0 should be aware of.

My publications for Scratch v2.0

On the release date for v2.0 (*May 9, 2013*) , I was about two-thirds complete in the publication of a series of modules designed to connect the *fun* side of Scratch v1.4 to the *serious* world of Computer Science. I immediately put that v1.4 series on the back burner and began a new series built around [Scratch v2.0](#). However, for the benefit of those who still need information about v1.4, I did not delete the modules that I had previously published for that version. You will find those modules in this collection in addition to the modules for v2.0.

Are you too old for scratch ??

If you are an "older" beginner, don't be put off by the fact that the Scratch forums are primarily attended by younger students. Scratch incorporates a wealth of information that is equally beneficial to both younger and older beginners. The main difference is that the older beginners are more likely to graduate from Scratch into a programming environment that is more commercially viable sooner than the younger students.

Additional drag-and-drop programming resources

If you would like to study some more advanced drag-and-drop material after you finish studying the Scratch modules, you might want to take a look at the following:

- [Snap! \(Build Your Own Blocks\)](#)
- [Blockly](#).

- [MIT App Inventor](#)

Snap! is an extended re-implementation of Scratch with many advanced features. It is used in computer science coursework at the University of California at Berkeley.

Blockly and the MIT App Inventor are products that attempt to bridge the gap between the use of drag-and-drop programming for education and the use of drag-and-drop programming for the development of commercial applications.

In addition, there are a number of other educational products available on the web that appear to be based on Scratch.

Text based learning resources

When you are ready for some text-based programming instruction, I have several recommendations for you in my order of preference:

1. [Codecademy JavaScript](#)
2. [Codecademy Ruby](#)
3. [Codecademy Web Fundamentals](#)
4. [Udacity - Introduction to Computer Science](#)
5. [Object-Oriented Programming.\(OOP\) with Java](#)

These are all free online courses.

It isn't necessary for you to establish an account to take the Codecademy courses, but there are some benefits to establishing an account. For example, with the Codecademy courses, if you establish an account and log in before working on a course, the system will keep track of where you are, what you have completed, etc. and will make it possible for you to resume where you left off the next time you log in.

Codecademy JavaScript

The first two items in the above list are almost tied in terms of my order of preference. However, the JavaScript course edges out the Ruby course for several reasons.

First, JavaScript is a ubiquitous programming language that is used for almost all interactive web pages. As such, it is a mainstay of the coding world.

Next, JavaScript is the most commonly used programming language in the rapid advance of HTML 5. Some of the features of HTML 5 are expected to eventually replace the need for the Adobe Flash player mentioned earlier with regard to Scratch 2.0.

Although JavaScript syntax is a little more complicated than Ruby syntax, JavaScript syntax is much closer to the syntax of major programming languages such as C, C++, C#, Java, and possibly others as well. The syntax began with C many years ago and has propagated forward into the newer and more powerful programming languages. Once a student understands this syntax, that student will be well equipped to deal with the syntax of other languages such as Ruby.

Finally, another reason that I prefer JavaScript over Ruby for teaching purposes is that JavaScript is very easy to experiment with after completing the course. All that is required is a browser and a text editor.

For example, if you use a text editor to copy the text shown below into a plain text file with an extension of *htm* or *html* , and then open that file in your browser, the JavaScript code between the beginning and ending *script* tags will be executed.

Note:

```
<html>
<body>
<script LANGUAGE="Javascript1.8">
```

```
<!-- Comment: put JavaScript code here -->
document.write("Hello from JavaScript")

</script>
</body>
</html>
```

The text shown above will cause the following text to appear in the browser window:

Hello from JavaScript

Codecademy Web Fundamentals

This course will teach you how to code HTML and CSS. HTML and CSS are not programming languages per se, but they do fall in the category of coding. Virtually every web page is based on HTML and web pages are often made more attractive through the use of CSS. Therefore, every coder needs to know how to code using HTML and CSS.

Like JavaScript, HTML and CSS are very easy to experiment with after completing the course. All that is required is a browser and a text editor. Simply create your HTML and CSS files using a text editor and then open the HTML file in your browser to see the results.

Codecademy Ruby

While JavaScript is the ubiquitous programming language for interactive web pages, Ruby is fast becoming one of the most popular programming languages for large-scale commercial websites. Those students who plan to go in that direction will do well to learn to program using the Ruby programming language.

Unlike JavaScript, some effort is required to get your computer set up to experiment with Ruby after you finish the course. You will probably need to do some web research to learn exactly how to do that.

Codecademy PHP and Python

The curious among you may wonder why I didn't include the Codecademy PHP and Python courses in my list of preferences. The answer is simple. As of this writing, I haven't had time available to work through and evaluate those courses.

Udacity - Introduction to Computer Science

The curious among you may also wonder why this is near the bottom of my list of preferred courses. I put it there largely because it is an interactive video course and I don't consider video to be a very effective way to learn how to code.

This is a free course in computer science intended for beginners. You must establish an account to take the course, but as of March 2013 the course is free. Having an account is an advantage. If you need to log out and do something else before completing a lesson, the next time you log in, the Udacity system allows you to resume at the point where you left off earlier.

Online video with interactive quizzes

This course makes an interesting and relatively seamless use of video content with interactive quizzes, but it is not for younger learners. It moves at a fairly rapid pace and I have noticed that sometimes the author includes material on the quizzes that has not yet been discussed in the videos. Don't be too concerned, therefore, if you are unable to answer the questions on the quizzes the first time through.

Multiple lessons

The course is divided into ten lessons plus some additional material. The first couple of lessons could be very useful to beginning programmers because they present some important fundamental programming concepts using the Python programming language.

The first lesson also contains some fairly abstract *Backus-Naur* material that seems to be out of place in an introductory lesson. Don't be too concerned if you don't fully understand that material. While critical for "computer science" students, you can go a long way in programming without fully understanding this material. *(It is probably more important for people who design programming languages than for people who use programming languages.)*

Beyond the first couple of lessons, the course gets into some fairly advanced material. However, if you understand the material in the first couple of lessons, don't hesitate to continue with the additional lessons.

Object-Oriented Programming (OOP) with Java

This website is included in the last three items in my [recommended path](#). The material on this website ranges from fundamental programming concepts for beginners to more advanced programming concepts for those with programming experience. This material includes the primary learning resources for several courses that I teach at Austin Community College.

Other options

Information about other options listed [above](#) will be added later.

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Tkc0100: Preface
- File: Tkc0100.htm
- Published: 03/17/13
- Revised: 05/30/13

Note: Disclaimers:

Financial : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.

I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module. In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.

Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

Appendix A: Programming knowledge requirements in Texas

Several years ago, I extracted information from various Texas State documents in an attempt to get a handle on the minimum level of programming knowledge that is required for students to graduate from a Texas high school with one or more courses in computer science.

A list of requirements

By organizing that material, I came up with the following as the minimum list of items that must be understood by the graduating student:

1. General knowledge of computer programming such as the ability to differentiate among the levels of programming languages including machine, assembly, high-level compiled and interpreted languages.
2. Problem-solving strategies such as design specifications, modular top-down design, step-wise refinement, and algorithm development.
3. Visual organizers to design solutions such as flowcharts or schematic drawings.
4. Variables
5. Expressions and Operators
6. Sequence, Selection, and Loop Structures
7. Relational and Logical Operators
8. Data types
9. Pretest and post-test loops.
10. One-Dimensional Arrays
11. Text files and structures of records.
12. Sequential search algorithms.
13. Use of simple data structures such as stacks and queues.
14. Use of the Java class library.
15. Command-line user input based on simple menus.
16. Coding proficiency in Java.
17. General Programming Syntax, Code Blocks, Comments
18. Writing and Calling Functions
19. Passing Function Parameters
20. Types of Errors (syntax versus logic)

-end-

Scr0300: Scratch 2.0 Overview

The purpose of this module is to provide a high-level overview of the Scratch 2.0 programming environment.

Table of Contents

- [Preface](#)
 - [General](#)
 - [A prediction](#)
 - [Where are the little leagues of computer science?](#)
 - [Some statistics](#)
 - [Many archived projects are readily available](#)
 - [Post, review, and comment](#)
 - [A world-wide phenomenon](#)
 - [Demographics](#)
 - [Not in organized classes](#)
 - [Viewing tip](#)
 - [Images](#)
- [Programming for many purposes](#)
 - [The Scratch \(Adobe Flash\).player](#)
 - [Computer art by pandalecteur](#)
 - [Anime projects](#)
 - [Game projects](#)
 - [3D games](#)
 - [A pseudo-3D animation](#)
 - [An animated story](#)
 - [The tip of the iceberg](#)
- [Scratch development environment](#)

- [A sprite-oriented programming environment](#)
- [Costumes](#)
 - [Walking-boy costumes](#)
- [The stage](#)
- [Controlling the behavior of a sprite](#)
- [Drag and drop programming](#)
- [Costumes and sounds](#)
- [The Scratch programming language](#)
 - [Music, music, music](#)
 - [The toolbox buttons and toolbox pane](#)
 - [High-level multimedia functionality](#)
 - [Program control](#)
- [Summary](#)
- [Resources](#)
- [Miscellaneous](#)

Preface

General

[Scratch 2.0](#) (*released May 9, 2013*) is the second major version of Scratch to be released during the life of the product. Among other things, it features a redesigned editor and website, and allows you to edit projects directly from your web browser.

This module is one in a collection of modules designed to help beginners of all ages (*8 and up*) learn how to create the code for computer programs using Scratch 2.0. Information is provided not only for the beginners themselves but also for their parents and teachers where appropriate.

The purpose of this module is to provide a high-level overview of the [Scratch](#) programming environment. Future modules will provide more

detailed information about the programming environment.

While this module is not intended to exclude the "older" beginners, much of the information contained in this module may be of more interest to the parents and teachers of younger students who are wondering if they should allow the students in their charge to become involved in Scratch.

A prediction

It's probably safe to say that a large percentage of the professional athletes in the major leagues of sports are people who became involved in sports in little league when they were quite young. It's probably also safe to say that a large percentage of successful professional musicians were either enrolled in serious musical instruction at an early age, or were involved as members of garage bands by their early teens.

I predict that in ten to fifteen years, computer science professionals and professionals in a variety of computer-related fields will be the people who are currently engaged in the *little leagues of computer science*.

Where are the little leagues of computer science?

One of the most important *little leagues of computer science* is centered in the [Lifelong Kindergarten group](#) at the [MIT Media Lab](#). This group has developed and is supporting a programming environment for beginners called [Scratch](#). According to MIT,

"Scratch is a programming language that makes it easy to create your own interactive stories, animations, games, music, and art -- and share your creations on the web.

Scratch is designed to help young people (ages 8 and up) develop 21st century learning skills. As they create Scratch projects,

young people learn important mathematical and computational ideas, while also gaining a deeper understanding of the process of design."

Scratch is available free of charge. Because Scratch 2.0 runs in a browser, it is compatible with all computers that support a compatible browser and the [Adobe Flash Player](#).

Some statistics

As of March 15, 2013, the Scratch home page indicated that 3,171,004 projects had been created and published on the Scratch website since inception. That number was increasing by about 230 projects per hour or 4 projects per minute at that time.

According to an [article](#) published by MIT personnel

"On the morning of May 14, 2007, the (Scratch Online Community) website was officially launched. Several news outlets and social news websites featured the Scratch website on their front pages. In a matter of hours the server and the website could not handle the traffic and the website went down several times."

If I did the arithmetic correctly, this represents an average of one new project every minute being posted on the website since the inception of the website in 2007. Of course, many other projects have been created but not published on the website.

Many archived projects are readily available

Unlike with v1.4, it is no longer necessary with v2.0 to download a project to examine the source code. Thousands of archived projects are available for online execution and examination at the source code level online.

*When you view a project online, a blue button in the upper-right corner of your screen labeled **See inside** will show you the source code for the project.*

Thus, the amount of resource material available to budding Scratch programmers is almost limitless.

Post, review, and comment

MIT makes it possible for every *scratcher* (as the members of the Scratch community refer to themselves) to share his or her projects for online execution and examination by other scratchers. (Project files can also be downloaded, but that capability is intended mainly for advanced users.) MIT also makes it possible for other scratchers to review and critique the projects shared by others. Registration is free and open to everyone.

MIT seems to make a significant effort to hide the true identities of the registrants and also seems to make a significant effort to ensure that the posted material is age appropriate for middle school students. (See [Community Guidelines](#).)

An organized system for peer review and critique is provided. The system is too elaborate for me to describe. If you want to know more about it, simply register (*become a scratcher*) and take a look at it for yourself.

A world-wide phenomenon

This is not just a United States phenomenon. The scratchers are located in many different countries around the world. A forum is provided for

scratchers to exchange information with one another. As of this writing, that forum is available in at least fifteen different languages.

Also, many scratchers whose first language is not English participate in the English version of the forums and provide English-language descriptions of their projects when they publish them.

Demographics

An article published around 2008 stated that based on the first five months of usage data,

"...users are primarily age 8 to 17, with a peak at age 12. A good number of users are adult computer hobbyists and educators that create projects in Scratch, even though a lot of them know other professional programming languages. Some members of the community have emerged as mentors that help the beginners and provide advice."

The article also stated,

"While 70 percent of users are male, no correlation was found between gender and the number of projects... This indicates that even though the majority of users are male, the females are as engaged in creating projects as the males."

Not in organized classes

By following the discussions in the forums, I have observed that a large percentage of scratchers are not enrolled in any sort of formal programming classes. Rather, this is something that they are doing on their own. In fact, it seems that for many scratchers, creating, sharing, and discussing projects is a form of social networking much like other students might engage in using the better known social networking sites.

Viewing tip

I recommend that you open another copy of this document in a separate browser window and use the following links to easily find and view the images while you are reading about them.

Images

- [Image 1](#). Rushes with sun by pandalecteur.
- [Image 2](#). How to draw a head by dialga..
- [Image 3](#). Marble Racer SBE 4000 by jamie.
- [Image 4](#). Perspective03 by dbal.
- [Image 5](#). Day Dream by cremeglance.
- [Image 6](#). Screen shot of the Scratch 2.0 development environment.
- [Image 7](#). Costumes for a boy walking.
- [Image 8](#). The toolbox buttons and toolbox pane.

Programming for many purposes

The Scratch (Adobe Flash) player

Scratch projects are normally executed within the Scratch development environment shown in [Image 6](#). Unlike the earlier version 1.4, you do not need to have the Scratch software installed on your computer. Instead all Scratch 2.0 projects are developed, executed, and viewed online in your browser. However, you must have the Adobe Flash Player installed on your computer to develop, execute and view the projects.

You can start a [project](#) running by clicking the green flag when the project finishes loading into your browser. You can stop the execution of a project by clicking the red button in the upper-right corner. You can restart the execution by clicking the green flag in the upper-right corner.

Computer art by pandalecteur

The scratchers who create and post these projects come at it from many different viewpoints, but they are all programming in order to accomplish their own purposes. My guess is that in many cases, programming is simply a means to an end. For example, I would categorize the scratcher named **pandalecteur** , whose work you will see in [Image 1](#) , as a serious artist. Image 1. Rushes with sun by pandalecteur.

Figure

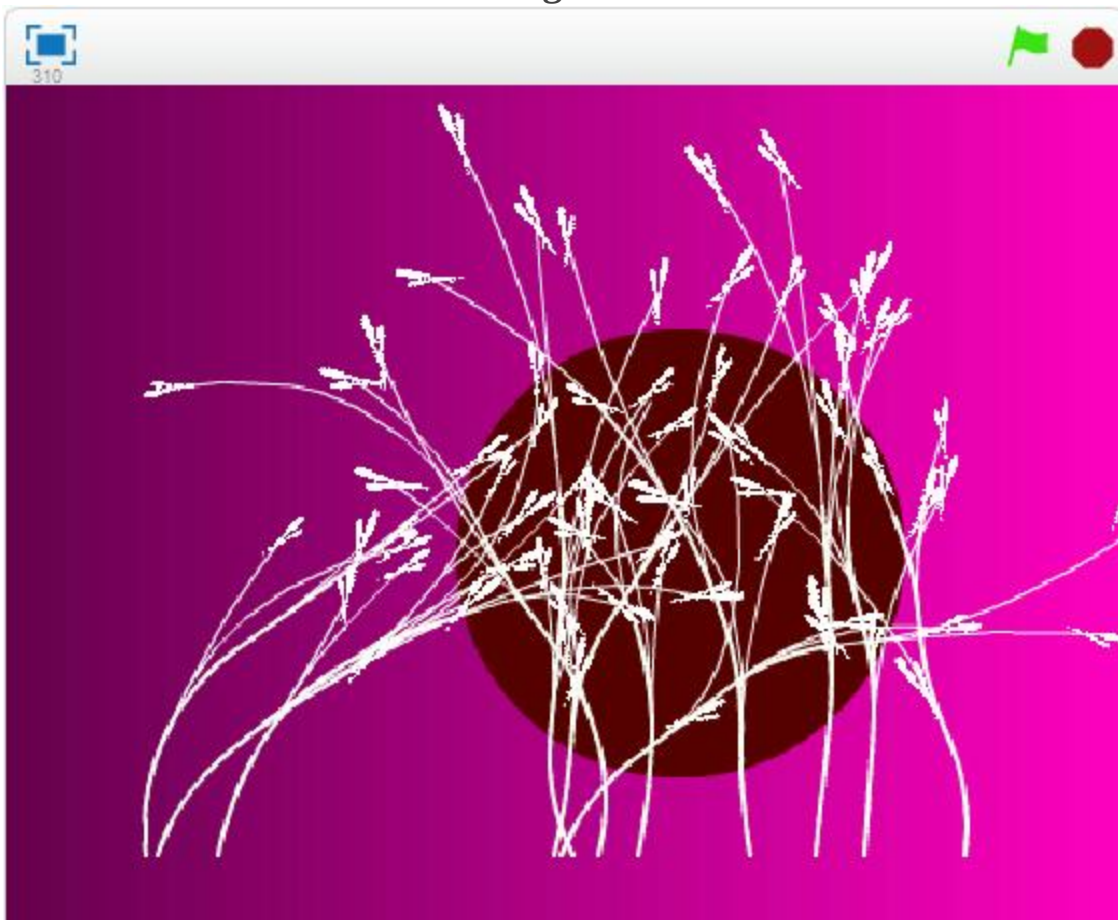


Image 1. Rushes with sun by pandalecteur.

[Image 1](#) contains a screen shot from the running [project](#) named *rushes with sun* . You will find links to more of her creations [here](#) . (*Just click on one of her projects and then click the green flag to start it running.*) She combines her programming talent with her artistic talent to produce beautiful animated creations.

Anime projects

[Image 2](#) represents another category of Scratch projects commonly seen on the Scratch web site. Many of the scratchers create the characters for what amounts to animated comic strips often referred to as *anime* . Sometimes the characters are used to tell a story and sometimes the programmers/authors/artists are content to simply draw and display the characters. Sometimes I read about two scratchers getting together with one doing the art work and the other doing the animation programming.

Image 2. How to draw a head by dialga.

Figure

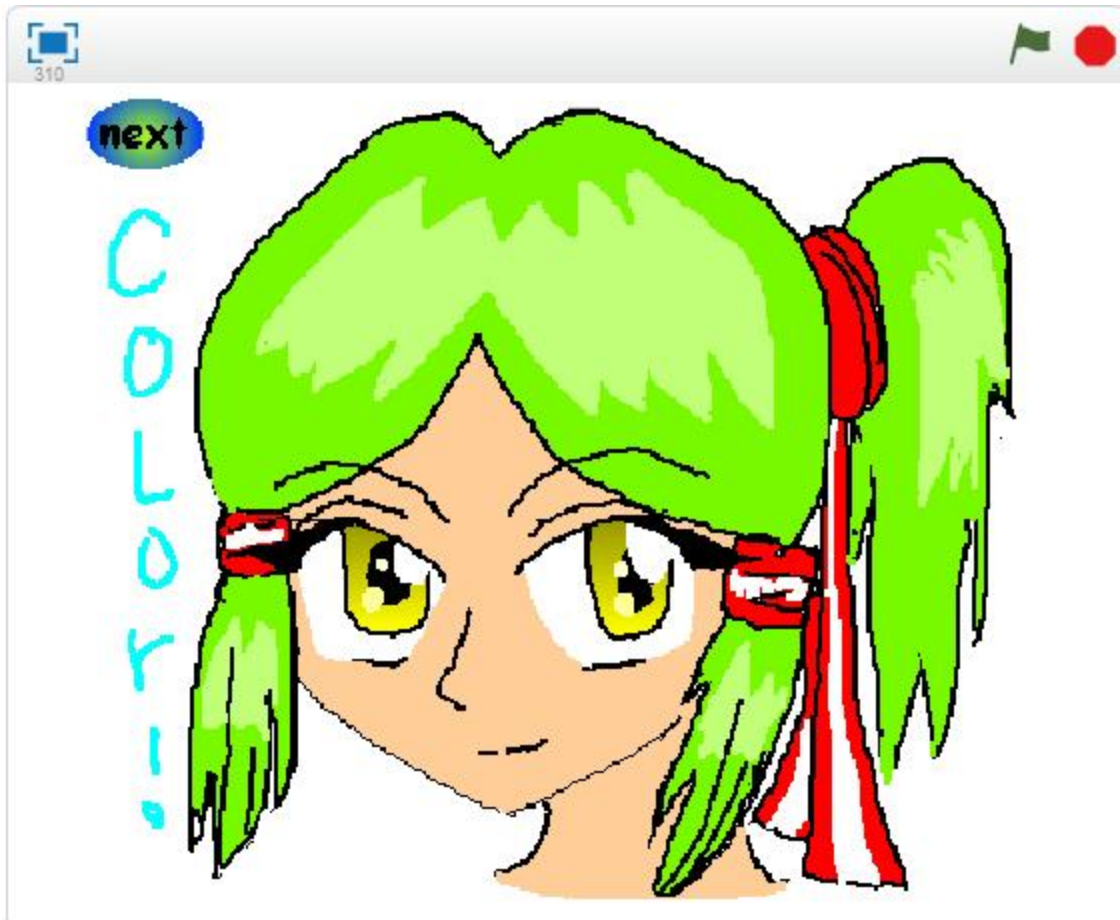


Image 2. How to draw a head by dialga.

[Image 2](#) shows an educational animation with instructions on how to draw the head of an anime character. Click [here](#) to view this project online in your browser. If you run the project and click on the green button labeled **next** in the upper-left corner several times in succession, the program will cycle through the steps involved in drawing a head.

I described **pandalecteur** earlier as a *serious artist* . In the same vein, I would describe **dialga** who created the project shown in [Image 2](#) as an *anime artist* . I'm not discrediting or indicating a preference for either artist. I am simply recognizing the different viewpoints of the two. Click [here](#) to view more of **dialga**'s creations.

Game projects

As you may have guessed, many scratchers like to create games. Most of them are 2D games such as the Marble Racer game by **jamie** shown in [Image 3](#). The object of the game is to use the arrow keys to cause the marble to roll all the way around the track without being slowed down by rolling onto the grass. Click [here](#) to view the project online. Click [here](#) to see more projects by **jamie**.

Image 3. Marble Racer SBE 4000 by jamie.

Figure



Image 3. Marble Racer SBE 4000 by jamie.

3D games

Although the mathematics involved are probably too advanced for most scratchers, a few scratchers push the envelope and attempt to make 3D games. My observation so far is that most of the 3D games that they make are so slow as to be extremely boring. Nonetheless, working on a 3D game in Scratch is a very educational process since Scratch doesn't provide any built-in support for doing 3D projections other than providing the necessary trigonometric functions.

A pseudo-3D animation

[Image 4](#) shows a pseudo-3D animation of my own design. In this project, the walking sprite follows the mouse pointer. Various tricks were employed to create an illusion of perspective in order to create an illusion of 3D. Click [here](#) to view the project in your browser.

Image 4. Perspective03 by dbal.

Figure

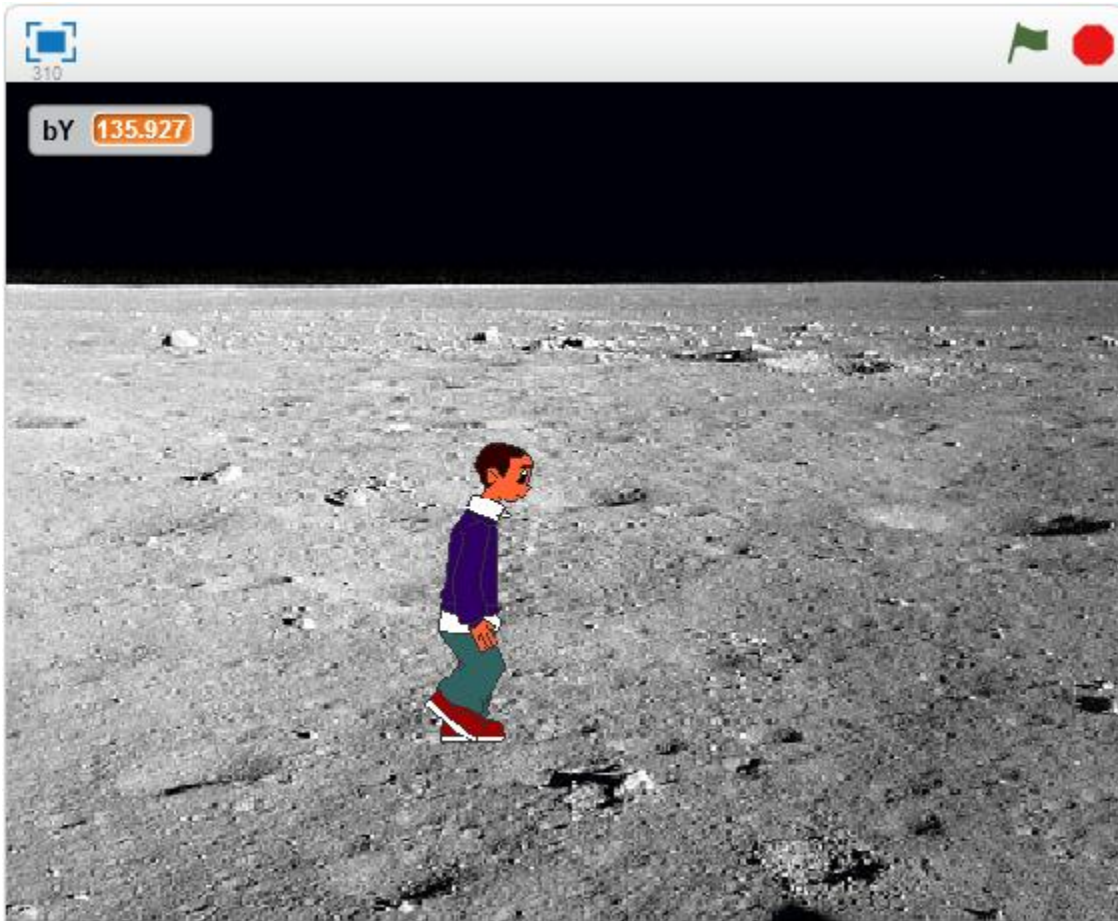


Image 4. Perspective03 by dbal.

An animated story

[Image 5](#) shows an animated story named **Day Dream** by **cremeglace**. In this project, the main character has a dream that goes through several scenes before reaching the end of the dream. The project includes music and dancing sprites. Click [here](#) to view the project in your browser.

Image 5. Day Dream by cremeglance.

Figure

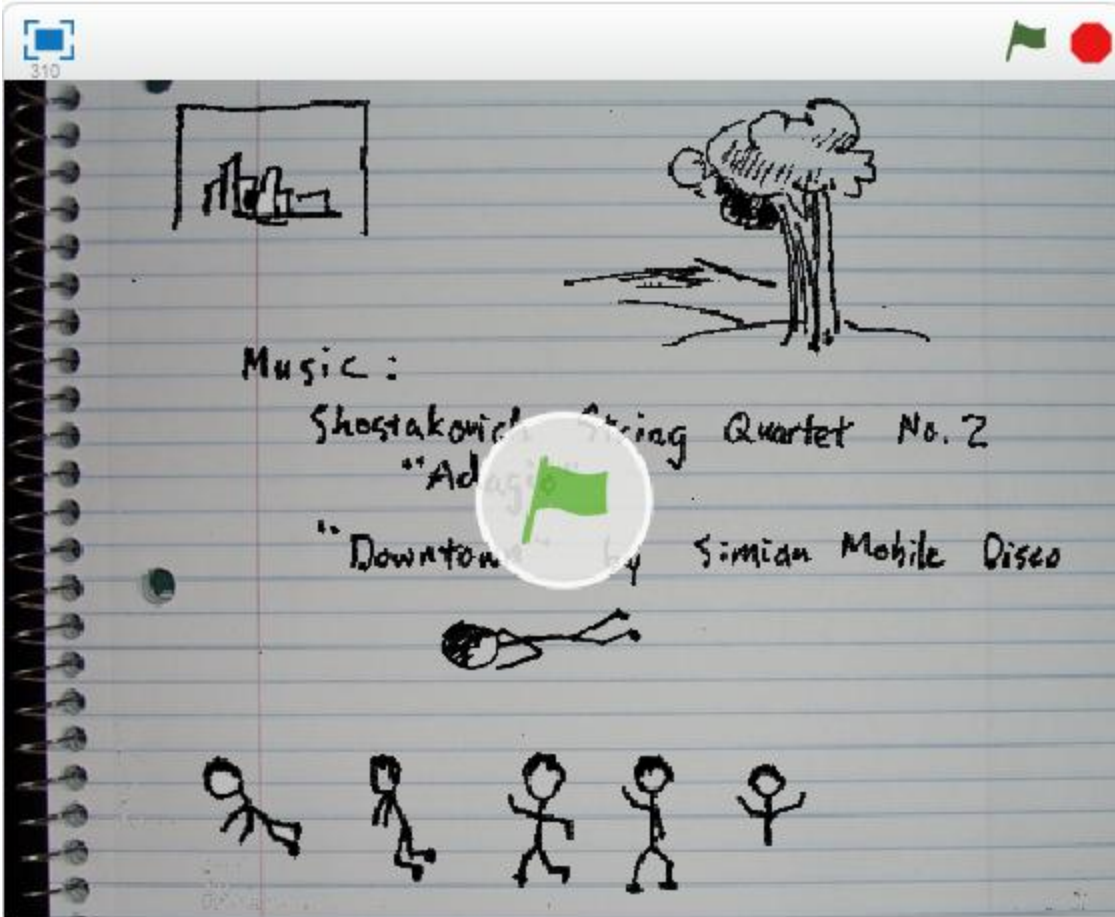


Image 5. Day Dream by cremeglance.

The tip of the iceberg

These five example projects are simply the tip of the iceberg in terms of the different purposes for which scratchers create projects. If you can imagine it in an animated computer program, some scratcher has probably already tried to do it.

Scratch development environment

[Image 6](#) shows a reduced screen shot of the Scratch 2.0 development environment. The text in [Image 6](#) is too small to read in most cases. The main purpose of [Image 6](#) is to show you the overall layout of the Scratch development environment as it appears in your browser.

Image 6. Screen shot of the Scratch 2.0 development environment.

Figure



Image 6. Screen shot of the Scratch 2.0 development environment.

A sprite-oriented programming environment

Scratch is a *sprite-oriented* programming environment. Every Scratch project consists of none, one, or more sprites and a stage upon which the sprites perform the behavior for which they are designed.

A sprite is an invisible entity to which the programmer can assign numerous behaviors. Each sprite can have one or more costumes. The assignment of a visible costume to an invisible sprite causes the sprite to also become visible.

Costumes

The Scratch development environment provides a large gallery of costumes and you can create your own costumes if you don't find any in the gallery that suit your needs. A costume is nothing more than an image. The background in the image is normally transparent. You can import images from a variety of different file types to create costumes. Probably the most difficult task in creating a costume from an imported image is causing the background to be transparent (*if the image didn't originally have a transparent background*). A built-in paint program is provided to assist in this task. You can also use programs such as the free [GIMP](#) image editor to deal with the background.

Note that one of the new features in v2.0 is to use both bitmap graphics and vector graphics. I will have more to say about this in a future module.

Walking-boy costumes

The project shown in [Image 4](#) uses five costumes from a family of seven costumes that are available in the gallery. Six of the seven costumes are shown in [Image 7](#). The project shown in [Image 4](#) uses only the five rightmost costumes shown in [Image 7](#).

Image 7. Costumes for a boy walking.

Figure

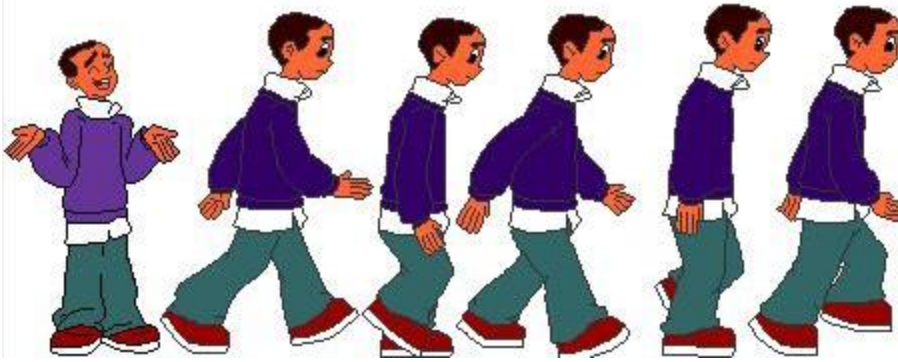


Image 7. Costumes for a boy walking.

As you can see, the five rightmost costumes describe five stages in a boy walking. The project shown in [Image 4](#) cycles through the five costumes in such a way as to make it appear that the boy is walking. (*This process is often called frame animation and dates back to or before the earliest days of the Disney productions.*)

The stage

The stage is the rectangular area showing the cat in [Image 6](#). Different images can be assigned to the stage to serve as backgrounds for the performance of the sprites. The gallery contains a large number of background images, and you can also create your own. (*One of those backgrounds is shown in [Image 6](#).*) Usually for a background image, you don't have the transparency issue discussed earlier so backgrounds for the stage can be easier to create than costumes for sprites.

A moonscape background from the gallery was used for the project shown in [Image 4](#).

Controlling the behavior of a sprite

You control the behavior of a sprite by writing one or more scripts that belong to the sprite. In reality, each script is an event handler of sorts. Usually you will have one event handler that executes when the user clicks the green flag at the top right of the stage in [Image 6](#). *(In v2.0, a large green flag button appears when you open a project for viewing in your browser.)*

In addition to the green-flag script, you can also write scripts to handle mouse events, key events, and a special type of event that fires when one sprite broadcasts a message to one or more other sprites. Therefore, Scratch projects are not only sprite-based; they are also event-based. These are relatively advanced concepts in most programming environments. However, Scratch was designed in such a way as to make it easy to write event-handler scripts to control the behavior of sprites.

Drag and drop programming

Creating Scratch programs involves very little typing. Instead, scripts are created by constructing stacks of blocks, where each block imparts some specific behavior to a sprite or to the stage. For example, the project being developed in [Image 6](#) has a stage, *(which is always the case)* and has a single sprite wearing a costume for a cat. The stage is represented by the rectangular thumbnail image in the bottom left area of [Image 6](#). Any sprites that are added to the project would also appear as thumbnail images in that same area. I will refer to this area as the sprite list area.

The physical process for writing a script is as follows:

1. Select the stage or a sprite in the sprite list area that is to be programmed.
2. Select the Scripts tab in the center pane.
3. Select one of the color-coded buttons in the toolbox button area at the top of the center pane to expose the blocks contained in a particular toolbox in the toolbox pane. The toolbox pane is the center pane currently showing the blue blocks in [Image 6](#).

4. Drag blocks from the toolbox to the scripts (*rightmost*) pane. (*Several blocks have already been dragged to the scripts pane in [Image 6](#).*)
5. Go back to 3 and continue this process until you have all of the blocks that you need in the scripts pane.
6. Snap the blocks together in the correct arrangement to create a script that produces the desired behavior.

You will learn the details of such operations in future modules.

Costumes and sounds

Selection of the other two tabs showing at the top of the center pane in [Image 6](#) exposes the controls for importing, editing, and creating costumes and backgrounds, as well as recording and/or importing sound files to be used as music and sound effects.

The Scratch programming language

Despite the fact that Scratch has amassed a huge following since its Beta release on March 4, 2007, in my opinion, version 1.4 of the language wasn't a particularly good programming language from a computer science viewpoint. Of the ten or fifteen programming concepts that most computer science professors consider to be fundamental to good programming, Scratch v1.4 supported only a few. Those few include:

- Variables and literals
- Sequence, selection, and loop structures
- Arithmetic, relational, and logical operators

These are very important concepts but notably absent was programmer-defined procedures, functions, or methods. I am happy to report that v2.0 seems to have resolved that deficiency.

Version 2.0 makes it possible for the programmer to create new blocks and to save those new blocks for use in ways that programmers in other

languages use procedures. I will have much more to say about this in a future module.

Another major addition to v2.0 from a computer science viewpoint is the ability for a sprite to create and/or delete a clone of itself at runtime. Therefore, it is no longer necessary for the programmer to know in advance how many enemy ships will appear on the horizon during game play. The number of enemy ships can depend on other factors at that point in the game play.

What Scratch v1.4 lacked in the support of fundamental programming concepts, it made up for in high-level multimedia functionality. In effect, Scratch v1.4 provided just enough in the way of fundamental programming capability to form a bridge to high-level multimedia functionality. The good news today is that v2.0 provides improvements in both fundamental programming concepts and high-level multimedia functionality.

Music, music, music

Among other things, *(particularly including the availability of a quasi social-networking community)* , it is probably this multimedia functionality that has attracted the large following among middle school and high school students.

There are a few projects on the website containing original music developed by the scratcher. *(I recall seeing one project where the scratcher was offering to compose original music to be used by other scratchers.)* However, that is the exception rather than the rule. Most of the music in the projects on the website appears to be copyright music for which the scratcher probably doesn't own the copyright.

That having been said, Scratch provides the capability for scratchers to compose their own music if they choose to do so.

The toolbox buttons and toolbox pane

[Image 8](#) shows a normal size screen shot of the center pane in [Image 6](#).
Image 8. The toolbox buttons and toolbox pane.

Figure

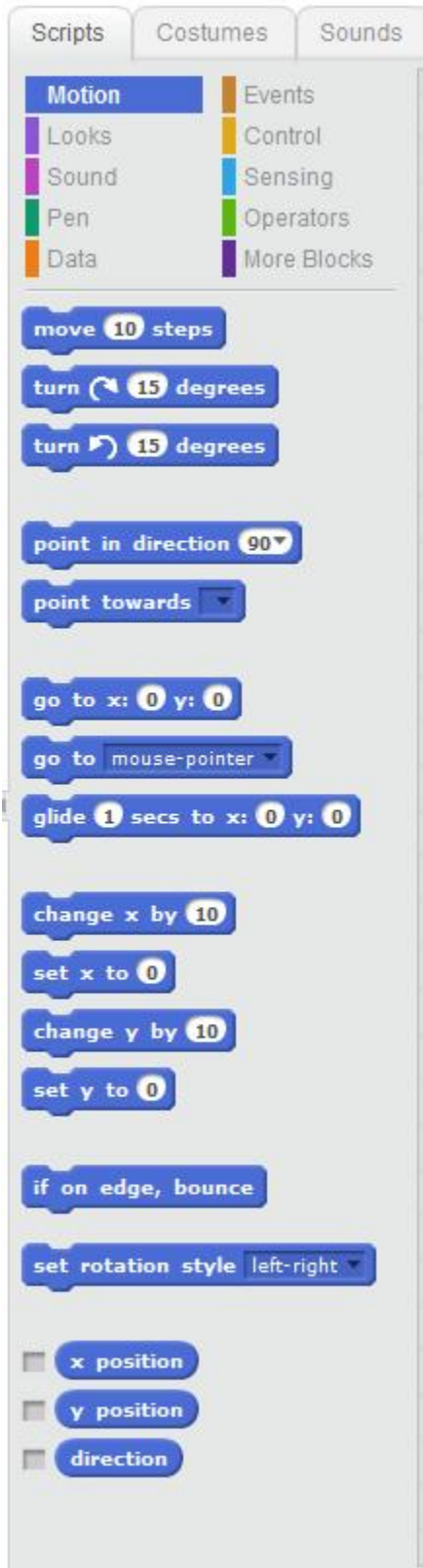


Image 8. The toolbox

buttons and toolbox
pane.

The small frame at the top of [Image 8](#) show the ten toolbox buttons that you click to expose a corresponding toolbox of blocks in the lower frame. (*The **Motion** toolbox was selected before capturing the screenshot shown in [Image 8](#).*)

High-level multimedia functionality

With regard to high-level multimedia functionality, Scratch makes it possible and relatively easy for scratchers to do the things they like to do such as:

- Create animation using blocks from the **Motion** toolbox.
- Deal with costumes, colors, graphics effects and sprite size using blocks from the **Looks** toolbox.
- Play imported music and sound effects and create original music and sound effects using blocks from the **Sound** toolbox.
- Create program-controlled line drawings using blocks from the **Pen** toolbox.

These capabilities, which are often difficult to access using "*typical CS-approved*" programming languages, provide a great deal of sensory feedback and they are a lot of fun to program. This is a large part of what causes Scratch to engage aspiring programmers.

Program control

This high-level multimedia functionality is controlled by:

- Programming event firing and detection, loop structures, and selection structures using blocks from the **Events** and **Control** toolboxes.
- Obtaining a variety of different types of information to be used in decision structures using blocks from the **Sensing** toolbox.
- Performing arithmetic, relational, and logical operations and accessing a variety of math functions using blocks from the **Operators** toolbox.
- Creating and servicing variables using blocks from the **Data** toolbox.
- Creating procedures, functions, methods, or whatever you choose to call them from the **More Blocks** toolbox.

As a practical matter, the top four buttons on the left in [Image 8](#) are the *fun* buttons and the remaining six buttons are the *control* buttons.

Summary

In this module, I have provided a high-level overview of the [Scratch v2.0](#) programming environment. Future modules will provide more detailed information about the programming environment.

Resources

Please be aware that many of the following links may take you to information about Scratch v1.4.

- [Scratch home](#)
- [Scratch tutorials](#)
- [Scratch explanatory video](#)
- [Scratch tutorials from MIT](#)
- [Scratch tutorial - Digital Art for All](#)
- [Scratch for Budding Computer Scientists](#)
- [Learn Scratch at learnscratch.org](#)
- [Scratch Tutorial - Space Shuttle Mission STS-2020](#)
- [SCRATCH TUTORIALS ICT In Primary Education 2012-2013](#)
- [Scratch Wiki](#)
- [Scratch WikiTable of ContentsWebsite - Scratch Wiki](#)
- [Scratch WikiTable of ContentsTutorials - Scratch Wiki](#)
- [Scratch WikiTable of ContentsProgram - Scratch Wiki](#)

- [Tutorial about Variables - Scratch Wiki](#)
- [Scratch support MIT web site](#)
- [Scratch resources at Scratch.ie](#)
- [ScratchEd Resources](#)
- [Scratch project Day Dream](#)
- [Scratch project Son of String Art](#)
- [Scratch Project Scratch Tutorial](#)

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Scr0300: Scratch 2.0 Overview
- File: Scr0300.htm
- Published:05/10/13

Note: Disclaimers:

Financial : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.

I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module. In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available

on cnx.org and that it was made and published without my prior knowledge.

Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Scr0310: Getting Started with Scratch 2.0

The purpose of this module is to get you started programming in Scratch 2.0.

Table of Contents

- [Preface](#)
 - [Let's have some fun in the process of learning](#)
 - [Fully interactive operating mode](#)
 - [Viewing tip](#)
 - [Images](#)
- [No computer setup required](#)
- [Screen size requirements](#)
- [Create a new project](#)
 - [Follow the steps](#)
 - [Examine the Tips](#)
 - [View the tutorials](#)
- [Will concentrate on the fundamentals](#)
- [Drag and drop programming](#)
 - [Mechanics are difficult to explain](#)
 - [Advantages and disadvantages](#)
- [Run some student projects](#)
- [Open the Help page](#)
- [What's next?](#)
- [Resources](#)
- [Miscellaneous](#)

Preface

[Scratch 2.0](#) (released May 9, 2013) is the second major version of Scratch to be released during the life of the product. Among other things, it features a redesigned editor and website, and allows you to edit projects directly from your web browser.

This module is one in a collection of modules designed to help beginners of all ages (*8 and up*) learn how to create the code for computer programs using [Scratch 2.0](#). Information is provided not only for the beginners themselves but also for their parents and teachers where appropriate.

The purpose of this module is to get you started programming in [Scratch 2.0](#).

Let's have some fun in the process of learning

Scratch was originally designed to be a good learning resource. It was also designed to help students learn to program and to have fun at the same time. After all, if you and/or your students are going to invest the time to learn how to write computer programs, you might as well have a little fun along the way.

I suggest that you go to the [Scratch home page](#) and the [Explore page](#) and click on a few projects to see what I mean. Just keep in mind that many of the projects that you will find there were written by youngsters and may or may not work as advertised.

One of my favorite Scratch projects is named [Son of String Art](#) (*hopefully it is still there*). You should be able to click on the little yellow shapes in the green buttons at the top of the viewing area and see your shape constructed with strings resembling the string art of the sixties.

Fully interactive operating mode

Note the use of the word *interactive* in the above header. What you will see when you select a Scratch project on the MIT website isn't simply a passive

embedded video like you might see at [YouTube](#) or [Vimeo](#). Instead it is a fully interactive program that allows you to control the program by clicking buttons, etc., provided of course that the program was written to be used in an interactive mode.

Viewing tip

I recommend that you open another copy of this module in a separate browser window and use the following links to easily find and view the images and listings while you are reading about them.

Images

- [Image 1](#). Step-by-Step Intro.
- [Image 2](#). The Tips pane.

No computer setup required

I explained quite a bit about Scratch in the earlier module titled [Scr0300: Scratch 2.0 Overview](#).

Assuming that you are able to access the [Scratch](#) website and run some of the projects, there is nothing more that you need to do to get your computer set up to create new projects in [Scratch 2.0](#).

However, if you can access the [Scratch](#) website but you are unable to run any of the projects, you may need to download and install the [Adobe Flash Player](#).

Screen size requirements

Quite a lot of space is needed for all of the items in the [project development screen](#). You probably need to have your browser set to full screen on a 1024 x 768 monitor in order to see everything without the requirement to

deal with horizontal and vertical scroll bars on the edges of the browser window.

For a 1024 x 768 monitor, you also need to click the little triangle immediately below and to the right of the stage to reduce the size of the stage and free up more space for other items.

Create a new project

Click the **Create** link at the top of the main Scratch web page to begin creating a new project. When you do, the **Step-by-Step Intro** pane shown in [Image 1](#) will probably appear on the right side of your browser window.

*(If the **Step-by-Step Intro** pane doesn't appear automatically, click the question mark on the far right side of the browser window to cause it or the **Tips** pane shown in [Image 2](#) to appear. If the **Tips** pane appears, select the **Step-by-Step Intro** link beneath the **Getting Started** label to get back to the pane shown in [Image 1](#).)*

Image 1. Step-by-Step Intro.

Figure



Image 1. Step-by-Step Intro.

Follow the steps

Follow the steps provided on the Step-by-Step Intro pane to create your first Scratch 2.0 project.

Examine the Tips

When you have completed your first project, click the **Tips** link at the top left of [Image 1](#) to expose the **Tips** pane shown in [Image 2](#). The links on the **Tips** pane will take you to a wealth of information regarding the use of Scratch 2.0.

When you are finished with the Tips pane, click the X-button in the upper-right corner of [Image 2](#) to dismiss it. You can get it back by clicking the question mark that will appear on the right edge of the browser window.

Image 2. The Tips pane.

Figure

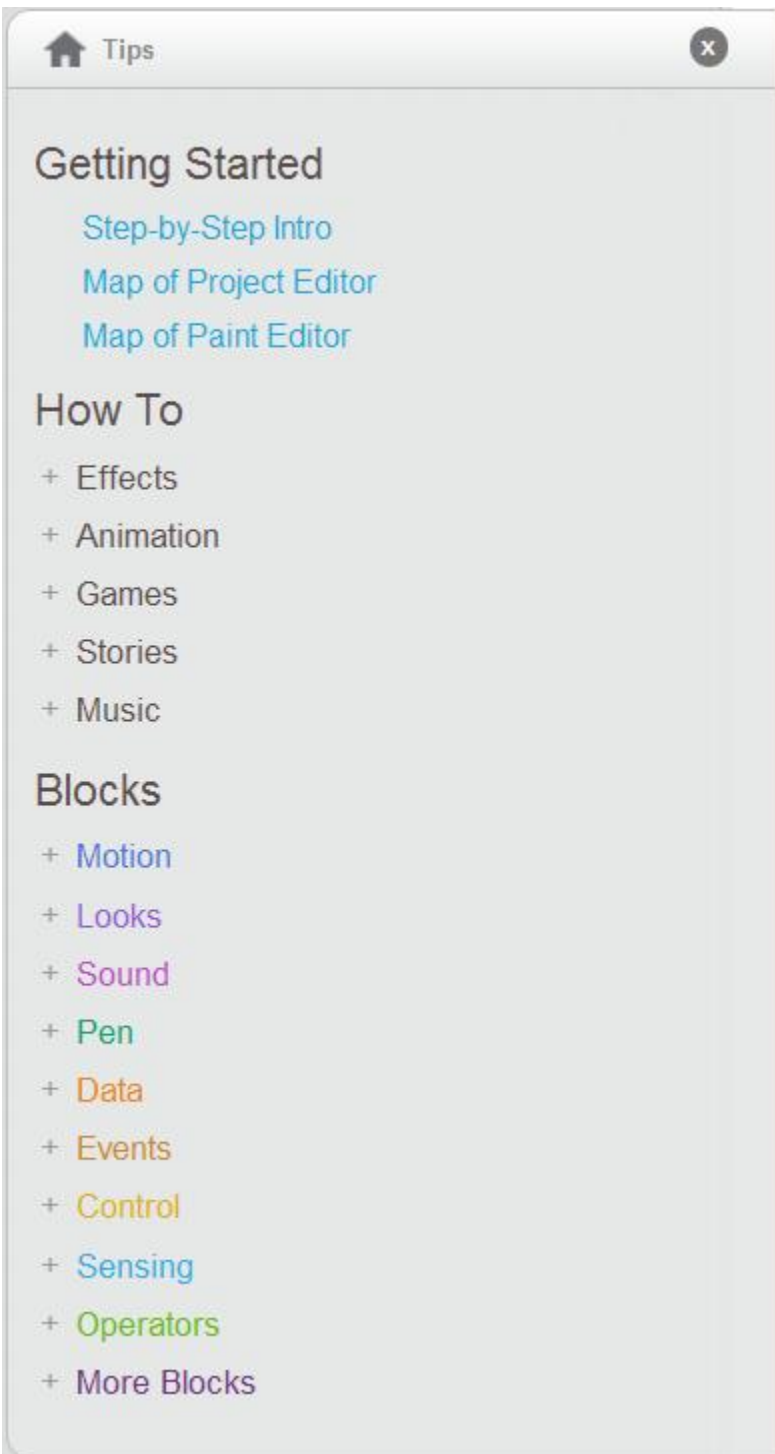


Image 2. The Tips pane.

View the tutorials

There are many tutorials scattered throughout the web explaining various aspects of programming with Scratch v1.4. Those tutorials are still useful because most of what you do to create Scratch projects hasn't changed with the release of Scratch 2.0.

Once you are comfortable with the material that you can reach from the links shown in [Image 1](#) and [Image 2](#), I recommend that you view every Scratch tutorial that you can find beginning with the tutorials at http://info.scratch.mit.edu/Video_Tutorials.

I have listed several tutorials in [Resources](#). Some are pretty good and some are not so good, but regardless, it would probably be useful for you to work through them all if you have the time available.

I also recommend that you run and pay particular attention to the [Scratch explanatory video](#) from MIT.

Will concentrate on the fundamentals

I doubt that you will find many (*or any*) Scratch tutorials that cover the fundamental programming concepts that I plan to cover in these modules. Most of the available Scratch tutorials tend to ignore the fundamentals and jump immediately into the higher-level features of Scratch such as playing music or viewing an image through a fisheye lens. That's not all bad. It is those features that are most likely to keep today's students engaged long enough for them to learn the fundamentals and go on to bigger and better things.

The modules in this collection will concentrate on the fundamental concepts of computer programming as identified in the earlier module titled [Tk0100: Preface](#). I view these modules as helping to build a bridge between the *fun* side of Scratch and the *serious* side of Computer Science.

Those fundamental concepts are transferable from one programming language to the next, and are required for learning the more advanced aspects of computer science.

The higher-level features of Scratch are not necessarily transferable to other languages. What I mean by that is that just because you know how to compose and play music in Scratch doesn't mean that you know how to compose and play music in some other programming language such as C++ or Java. However, if you understand the fundamental programming concepts embodied in Scratch, you should be able to transfer that understanding to other programming languages.

Drag and drop programming

One of the reasons that it will be useful for you to work through the tutorials is to help you gain proficiency in the *drag and drop* programming paradigm used with Scratch. This is a relatively new programming paradigm, and it is very different from the paradigm used for *old school* languages such as C, C++, C#, and Java.

"It seems strange to speak of Java as an old school programming language, but it has been a viable programming language for a little more than fifteen years now. In today's fast paced technology world, that probably makes it old school."

Mechanics are difficult to explain

Although drag and drop programming has some major advantages over the old paradigm for the beginning programming student, the mechanics involved are much more difficult to explain in a written tutorial than the mechanics of programming with Java, for example.

With the old school paradigm, all that is necessary to create a program is to:

- Open a text editor and type in the program.
- Open a compiler and compile the program.
- Execute the program.

Advantages and disadvantages

Because everything in old school programming is based on text, it is easy to explain it in a text-based written tutorial. The disadvantage of old school programming insofar as the student is concerned is that the student is required to memorize difficult and sometimes arcane programming syntax. This requirement is the downfall of many aspiring programming students.

The advantage of drag and drop programming is that the student is not required to memorize programming syntax; at least not in the beginning. That leaves the student free to concentrate on concepts.

One disadvantage of drag and drop programming is that the mechanics are very difficult to explain in a written tutorial. It is sort of like trying to explain to someone how to tie their shoes in a written tutorial. You almost need to see it done in order to learn how to do it yourself.

The best way to explain many aspects of drag and drop programming is through "*show me*" videos.

Many of the available tutorials are video tutorials, which make it possible for you to see how others accomplish drag and drop programming using Scratch. Once you see how it is done, you should have no difficulty doing it yourself.

Run some student projects

As I mentioned earlier, I also recommend that you run some of the [student projects](#) just to see what students are doing with Scratch. If you encounter problems running the student projects, you might consider seeking help on the Scratch [forums](#).

Open the Help page

Once you have opened the Scratch web page in your browser, you should take a look at the **Help** material that is available by clicking the **Help** link at the top of the page. You will find links there to a variety of useful resources.

What's next?

In the next module, I will begin the process of helping you to learn about the following computer programming concepts using Scratch:

- Memory
- Variables
- Literals

Resources

Please be aware that many of the following links may take you to information about Scratch v1.4.

- [Scratch home](#)
- [Scratch tutorial](#)
- [Scratch explanatory video](#)
- [Scratch tutorials from MIT](#)
- [Scratch tutorial - Digital Art for All](#)
- [Scratch for Budding Computer Scientists](#)
- [Learn Scratch at learnscratch.org](#)
- [Scratch Tutorial - Space Shuttle Mission STS-2020](#)
- [SCRATCH TUTORIALS ICT In Primary Education 2012-2013](#)
- [Scratch Wiki](#)
- [Scratch WikiTable of ContentsWebsite - Scratch Wiki](#)
- [Scratch WikiTable of ContentsTutorials - Scratch Wiki](#)
- [Scratch WikiTable of ContentsProgram - Scratch Wiki](#)
- [Tutorial about Variables - Scratch Wiki](#)
- [Scratch support MIT web site](#)
- [Scratch resources at Scratch.ie](#)
- [ScratchEd Resources](#)
- [Scratch project Day Dream](#)
- [Scratch project Son of String Art](#)
- [Scratch Project Scratch Tutorial](#)
- [Discuss Scratch](#)

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Scr0310: Getting Started with Scratch 2.0
- File: Scr0310.htm
- Published: 05/10/13

Note: Disclaimers:

Financial : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.

I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module. In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.

Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Scr0320: Memory, Variables, and Literals in Scratch 2.0

The purpose of this module is to teach you about memory, variables, literals, and algorithms in Scratch 2.0. You will also learn how to write a Scratch program that illustrates the creation and use of variables, case sensitivity, and the ability to detect and respond to mouse and keyboard events.

Table of Contents

- [Preface](#)
 - [Student programming projects](#)
 - [Viewing tip](#)
 - [Images](#)
- [General background information](#)
 - [An overview of computer programming](#)
 - [Not especially difficult](#)
 - [One step at a time](#)
 - [The hard work is often done for us](#)
 - [Memory](#)
 - [Random access versus sequential memory](#)
 - [Combination random/sequential memory](#)
 - [Variables](#)
 - [Brief definition of a variable](#)
 - [A physical analogy](#)
 - [Pretend that you are a computer program](#)
 - [Names versus addresses](#)
 - [Execute an algorithm](#)
 - [Declaring a variable](#)

- [Literals](#)
 - [Where does data originate?](#)
 - [Hard coded data](#)
 - [A value coded into the program](#)
- [Preview of the Scratch program](#)
 - [The Scratch program named Variable01.sb](#)
 - [A variable named Counter \(upper-case C\)](#)
 - [A variable named counter \(lower-case c\)](#)
- [Discussion and sample code](#)
 - [No sprites](#)
 - [The toolbox buttons](#)
 - [The Data toolbox](#)
 - [Create two variables](#)
 - [Allowable operations for variables](#)
 - [Specifying the values](#)
 - [Cause the variables to be displayed](#)
 - [Create a new variable](#)
 - [Allowable variable names and lengths](#)
 - [Meaningful variable names](#)
 - [Writing the program](#)
 - [Programming blocks in the Events category](#)
 - [The finished program](#)
 - [Behavior of the program](#)
 - [Click the green flag](#)
 - [Press the space bar](#)
 - [Click the mouse on the Stage](#)
 - [Event-driven programming](#)

- [Example blocks with pointed ends](#)
- [Run the program](#)
- [Student programming projects](#)
 - [Project 1](#)
 - [Project 2](#)
- [Summary](#)
- [What's next?](#)
- [Resources](#)
- [Miscellaneous](#)

Preface

[Scratch 2.0](#) (*released May 9, 2013*) is the second major version of Scratch to be released during the life of the product. Among other things, it features a redesigned editor and website, and allows you to edit projects directly from your web browser.

This module is one in a collection of modules designed to help beginners of all ages (*8 and up*) learn how to create the code for computer programs using [Scratch 2.0](#). Information is provided not only for the beginners themselves but also for their parents and teachers where appropriate.

The purpose of this module is to teach you about memory, variables, literals, and algorithms in [Scratch 2.0](#). You will also learn how to write a Scratch program that illustrates the creation and use of variables, case sensitivity, and the ability to detect and respond to mouse and keyboard events.

Student programming projects

In addition to presenting and explaining a Scratch programming project, I will present two student programming projects that are designed to:

- Help the student retain the knowledge gained by studying the module.
- Require the student to think beyond the material presented in the module by requiring the student to answer the question "How do I ...?"

Viewing tip

I recommend that you open another copy of this module in a separate browser window and use the following links to easily find and view the images while you are reading about them.

Images

- [Image 1](#). Cropped image of the Scratch programming interface for the program named Variable01.
- [Image 2](#). Dialog for entering a variable name.
- [Image 3](#). Programming blocks in the Events category.
- [Image 4](#). Example blocks with pointed ends.
- [Image 5](#). Output from Project 1.
- [Image 6](#). Output from Project 1.
- [Image 7](#). Output from Project 2.
- [Image 8](#). Output from Project 2.

General background information

An overview of computer programming

I recognize that some of you may not be ready for the kind of technical detail that I will provide in this section. If so, feel free to skip ahead to the section titled [Preview of the Scratch program](#). Just remember that this material is here waiting for you to come back and study it when you are ready.

Not especially difficult

Computer programming is not especially difficult. However, it does require an aptitude for solving problems. In fact, a computer program is usually a model for a solution to someone's problem.

One step at a time

If you can learn to program a Digital Video Recorder (DVR), you can probably also learn to program a computer. Of course, a computer is more complicated than a DVR, so there is more to learn. The important thing is to take the learning process in deliberate steps making certain that you understand each step before moving on to the next one.

The hard work is often done for us

Fortunately, when we write programs using a high-level programming language such as Scratch, much of the hard work is done for us behind the scenes. For example, we don't have to perform the small incremental steps that are required to divide one number by another number.

As programmers, we are more like conductors than musicians. The various parts of the computer represent the musicians. We tell them what to play, and when to play it, and if we do our job well, we produce a solution to someone's problem.

Memory

All computers contain *memory* of one type or another. When we speak of human memory, we are usually speaking of the things that the human remembers. However, when we speak of computer memory, we are usually speaking of physical devices where data is stored for later retrieval. Most modern computers contain memory of a type that is often referred to as RAM (*more on RAM later*) .

All data is stored in a computer as numeric values. Computer programs do what they do by executing a series of *operations* on the numeric data.

Numeric data *Even the text in this module is stored as numeric data in your computer. For example, the upper-case character "A" is commonly represented by the numeric value 65.*

The operations that are performed on the numeric data generally consist of calculations and comparisons. It is the order and the pattern of those operations that distinguishes one computer program from another.

Random access versus sequential memory

The reason the memory in most modern computers is called *random access memory (RAM)* is that it can be accessed in any order. Some types of memory, such as a magnetic tape, can only be accessed in sequential order. *(Yes, I did use computers with magnetic tape memory in my younger days during the sixties and seventies.)*

Sequential access means that to get a piece of data that is stored deep inside the memory, it is necessary to start at the beginning and examine every piece of data until the correct one is found. This is typically a slow process.

Combination random/sequential memory

Other types of memory, such as disks provide a combination of sequential and random access. For example, the data on a disk is stored in tracks that form concentric circles on the disk. The individual tracks can be accessed in random order, but the data within a track must be accessed sequentially starting at a specific point on the track. While faster than magnetic tape, even this process is usually slower than true random access memory.

Sequential memory or combination random/sequential memory isn't very useful for most computer programs at the inner working level because access to each particular piece of data is slow. As a result, most modern computers have random access memory that is used for storage and retrieval of data by programs while they are running and disks that are used for long-term storage and retrieval of data that needs to be saved over longer periods of time.

Variables

As the computer program performs its operations in the prescribed order, it is often necessary for it to store intermediate results somewhere in its memory and to retrieve those results later for use in subsequent operations. The intermediate results are often stored in little chunks of memory that we refer to as *variables*.

Brief definition of a variable

The following definition is paraphrased from [Wikipedia](#).

*In computer programming, a **variable** is a storage location and an associated symbolic name (an identifier) that contains some known or unknown quantity or information -- a value.*

The variable name is the usual way to reference the stored value; this separation of name and content allows the name to be used independently of the exact information it represents.

The value stored in the variable may change during program execution.

A physical analogy

We can think of random access memory as being analogous to a metal rack containing a large number of compartments. The compartments are all the same size and are arranged in a regular grid of rows and columns.

Each compartment has a numeric address printed above it. No two compartments have the same numeric address. Each compartment also has a little slot into which you can insert a name or a label for the compartment. No two compartments can have the same name.

Although the analogy is not perfect, we can think of one of those compartments as being analogous to a variable.

Pretend that you are a computer program

Think of yourself as a computer program. You have the ability to create labels for each compartment (*variable*). You have the ability to write values on little slips of paper and to put them into the compartments. You also have the ability to read the values written on the little slips of paper and to use those values for some purpose. However, there are four rules that you must observe:

- You may not remove a slip of paper from a compartment without replacing it by another slip of paper on which you have written a value.
- You may not put a slip of paper in a compartment without removing the one already there.
- You may not give the same label or name to two or more compartments.

- You may not change a label once you have assigned it to a compartment.

Names versus addresses

Although each *compartment* in the physical memory in the computer has a numeric address, as a programmer using a *high-level* programming language such as Scratch, you usually don't need to be concerned about the numeric addresses of the compartments. (*The compartments are often referred to as locations in memory.*)

Instead, you can think about the compartments and refer to them in terms of their names. (*Names are easier for most people to remember and understand than numeric addresses.*) Keep in mind, however, that computer memory locations don't really have names. Deep inside the computer program, the names that you use to identify compartments in memory are cross-referenced to memory addresses. At the lowest level, the program works with memory addresses instead of names.

Execute an algorithm

A computer program always executes some sort of procedure or algorithm. The algorithm may be very simple (*such as how to make a peanut butter sandwich*) , or it may be very complex as would be the case for a spreadsheet program. As the program executes the algorithm, it uses the random access memory to store and retrieve the data that is needed to execute the algorithm.

Declaring a variable

In Scratch, it is necessary to establish the *name* of a variable before you can use it. (*That is not the case in some programming languages such as Python and JavaScript.*)

In programmer jargon, this is referred to as *declaring a variable* . The process of declaring a variable causes memory to be set aside to contain a value, and causes that chunk of memory to be given a name. That name can be used later to refer to the value stored in that chunk of memory.

Some programming languages, (not including Scratch), require that the variable declaration also specify the type of data that will be stored in a variable.

Case sensitivity

The name of a variable is case sensitive in Scratch (*but the name may not be case sensitive in other programming languages*) . In other words, the name **aVariable** is not the same as the name **avARIABLE** in case-sensitive programming languages. If both names were used in the same program, they would refer to two different variables.

Use names to access values

Variables typically have names like **price** , **cost** , and **markup** . The names of the variables give us easy access to the values stored in the variables.

Some operations cause the value of a variable to change while other operations simply access the value without changing it.

Storing and retrieving

As the program executes, the values stored in variables can be replaced by other values. In that way, the values of the variables can change as the program executes. (*In other words, the value of a variable can vary over time.*)

It is also possible to retrieve the value stored in a variable without modifying it. In that way, the values of variables can be used to evaluate expressions without modifying those values.

Literals

Where does data originate ?

The data used by a computer program can originate from a variety of sources. For example, it can be entered from the keyboard. It can be read from a disk file. It can be downloaded from a web site, etc.

Hard coded data

The data values can also be coded into the program when the program is written. In this case, we would call it a *literal* or a *literal value* .

A value coded into the program

That is just about all there is to understanding literals. There are some special rules that come into play in certain situations, and I will discuss those rules at the appropriate points in time. For now, just remember that a *literal* is a value that is coded into the program when the program is written.

Preview of the Scratch program

Now it's time to write and discuss a Scratch program. You should be able to replicate this program.

Hopefully by now you have worked your way through the **Step-by-Step Intro** and the various **tips** that you can access by clicking the question mark on the right side of your browser window when the window is open on the

Scratch website's [programming interface](#). (Click the **Create** link on the [Scratch website](#) to open the [programming interface](#).)

Hopefully you have also worked through several of the tutorials that you can find on Scratch as recommended in the previous module. If not, you should do that before continuing with this module. In other words, by now you should have experience moving Scratch blocks around to cause your sprites to behave as you want them to behave.

In case you skipped the section titled [An overview of computer programming](#), you might still want to go back and review the section on [Variables](#).

The Scratch program named Variable01 .sb

Click [here](#) to run this program in your browser.

This Scratch program illustrates the creation and manipulation of two simple *numeric* variables.

Scratch variables can contain numeric values or strings of characters but this program will deal only with numeric values.

This program also illustrates the case sensitivity of variable names along with the ability to detect and respond to events fired by the keyboard and events fired by the mouse.

A variable named Counter (upper-case C)

A variable named **Counter** (with an upper-case "C") is created, initialized to 0, and displayed in the upper-left corner of the Stage area in the Scratch programming interface (see [Image 1](#)). Although the label on this variable

is small and difficult to read, you will see when you [run](#) this program that this is the topmost variable displayed on the Stage.

When the user clicks the green flag in the upper-right corner of the stage , the value of the variable is set to zero. Each time the user presses the space bar, the value of the variable is increased by 1. Each time the user clicks the mouse in the blank area of the Stage, the value of the variable is decreased by 1.

A variable named counter (lower-case c)

Another variable named **counter** (*with a lower-case "c"*) is created, initialized to 0, and displayed below the **Counter** variable on the Stage. Again, when the user clicks the green flag in the upper-right corner of the stage in [Image 1](#), the value of the variable is set to zero. Each time the user presses the space bar, the value of this variable is increased by 5. Each time the user clicks the mouse in the blank area of the Stage, the value of this variable is decreased by 5.

Discussion and sample code

[Image 1](#) shows a cropped image of the Scratch programming interface for the program named **Variable01** for Scratch v2.0. This image was produced by [running](#) the program online and clicking the blue button in the upper-right corner labeled **See inside** .

Image 1. Cropped image of the Scratch programming interface for the program named Variable01.

Figure

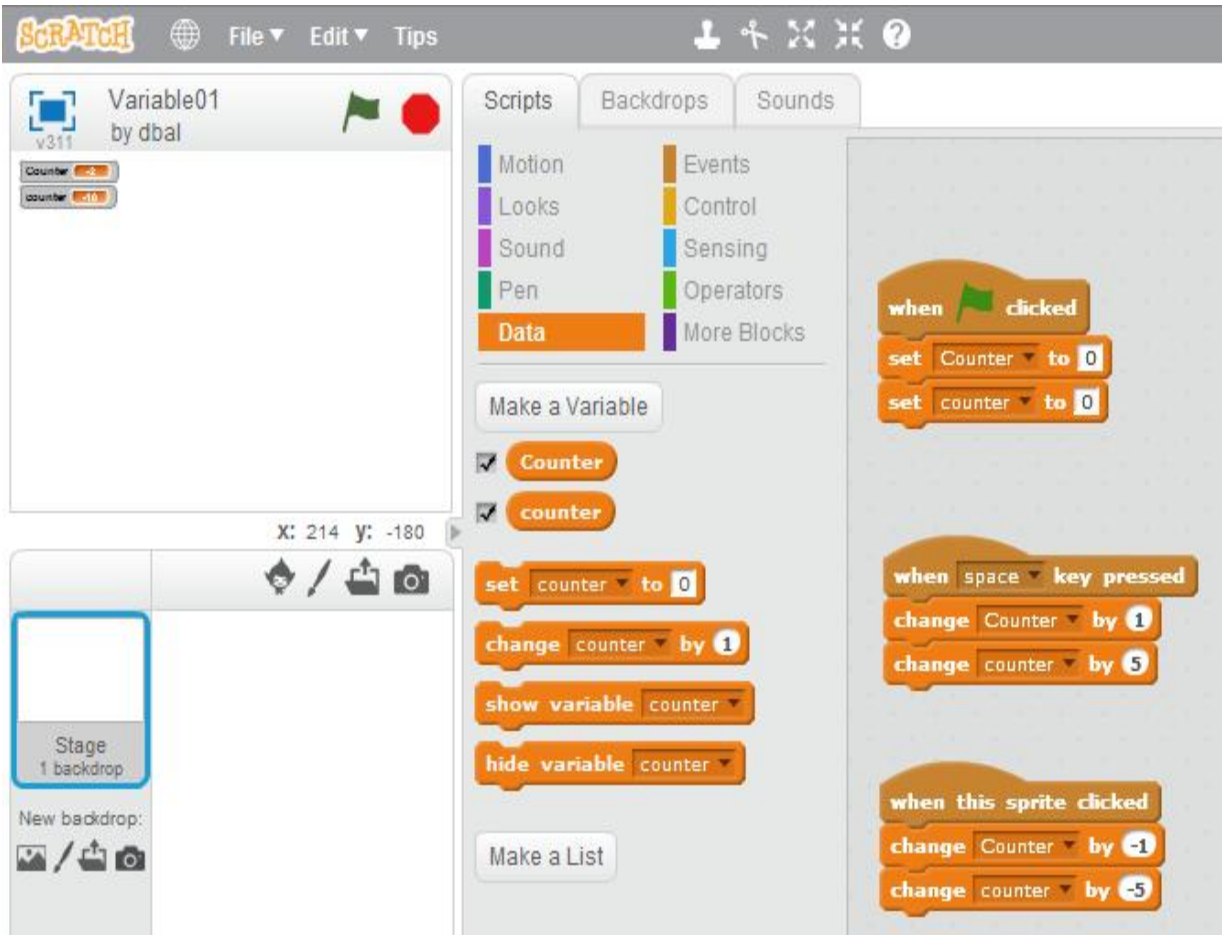


Image 1. Cropped image of the Scratch programming interface for the program named Variable01.

No sprites

There are a few things that we can conclude by viewing [Image 1](#). First, there are no sprites in this program. The program consists solely of

- The Stage, shown in white in the upper left,
- A small program consisting of three scripts shown in the rightmost panel,

- Two variables shown in orange near the middle of the center panel and also shown in gray/orange in the upper-left corner of the Stage.

The toolbox buttons

If you have worked through some Scratch v2.0 examples, you will recognize that the material in the lower-center panel is always associated with one of the following ten toolbox buttons **in the upper-center panel**:

- Motion
- Looks
- Sounds
- Pen
- Data
- Events
- Control
- Sensing
- Operators
- More Blocks

In other words, when you click on one of the ten buttons in the upper-center panel, it exposes a set of tools that you can use to write your program. Those tools appear in the lower-center panel.

When one of those ten buttons is clicked, it becomes completely colored and the tools associated with that button are displayed below it. The orange button labeled **Data** has been selected in [Image 1](#).

The Data toolbox

The lower-center panel of [Image 1](#) presents the Data toolbox showing the tools that are exposed by the Data button.

Tools in the Data toolbox: When you first click the **Data** button, only two gray buttons are exposed in the toolbox. One gray button is labeled **Make a variable** and the other gray button is labeled **Make a list**. The two orange variables and the orange tools shown in the **Data** toolbox of [Image 1](#) were produced by clicking twice on the button labeled **Make a variable** and entering a name for each variable.

Create two variables

Click [here](#) for detailed instructions on creating variables.

[Image 1](#) shows that this Scratch program contains two variables with the following names:

- **Counter**
- **counter**

I purposely spelled the names of the two variables the same and made them differ only by the case of the first letter to illustrate that Scratch is case sensitive insofar as the names of the variables is concerned. Even though these two variable names contain the same letters, they are two different variables because one name begins with an upper case "C" and the other name begins with a lower-case "c".

Allowable operations for variables

There are four programming blocks (in the **Data** toolbox of [Image 1](#)) associated with each variable. As you have probably learned from working through example programs, you can drag these blocks into the programming area in the rightmost pane in [Image 1](#) to actually write the program. Thus, there are four different operations that you can perform on a variable:

- You can set the value of the variable to a specified value.
- You can change the value of the variable by a specified value.
- You can show a variable on the stage.
- You can hide a variable.

(You can manually perform the last two operations by checking or clearing the checkboxes next to the variables but the bottom two tools allow you to perform that operation as part of a running program.)

Specifying the values

After dragging a programming block from the **Data** toolbox into the programming area, you can:

- Identify the variable to which that block applies by selecting the variable name in the pull-down list built into the block.
- Enter a literal numeric or string value from the keyboard into the white or gray text field at the right end of the block, or
- Drag some other block and drop it into the white or gray text field.

There are at least two different shapes that are eligible for being dropped into such text fields:

1. A horizontal rectangle with **rounded corners** such as the orange block labeled **Counter** in [Image 1](#). *(This shape seems to always be associated with a numeric or string value.)*
2. A horizontal rectangle with **pointed ends** as shown by several of the blocks in [Image 4](#). *(This shape seems to always be associated with a **boolean** result of true or false.)*

Blocks having the rounded corners are eligible for being dropped into the white text fields with rounded corners of the programming blocks in [Image 1](#).

Blocks having the pointed ends are eligible for being dropped into the darker orange text fields having a similar shape on the blocks that are exposed by clicking the **Control** button in the programming interface.

Either shape of block is eligible for being dropped into the white text fields with square corners shown in [Image 1](#). Thus the block in [Image 1](#) that is used to set a variable to a value can set the value to a numeric value, a string value, or a **boolean** value.

*(Note that the two variable blocks shown in [Image 1](#) labeled **Counter** and **counter** are eligible for being dropped into the white text fields of the two programming blocks shown in [Image 1](#). This makes it possible to cause the value of one variable to depend on the value of another variable.)*

*If you click the **Operators** button shown in [Image 1](#) (see [Image 4](#)) you will expose several different blocks with rounded corners that are eligible for being dropped into the white text fields in [Image 1](#) according to the rules described above.*

Cause the variables to be displayed

As mentioned earlier, checking the boxes to the left of the variable names in [Image 1](#) causes them to be displayed in the upper-left corner of the Stage shown in [Image 1](#). (Clicking the **show** and **hide** blocks will also check and uncheck the checkboxes, which in turn will show or hide the variables on the stage.)

Create a new variable

You create a new variable by clicking the button labeled **Make a variable** shown in [Image 1](#). When you click that button, the dialog shown in [Image 2](#) appears on the screen asking you to provide the name for the new variable.

Image 2. Dialog for entering a variable name.

Figure

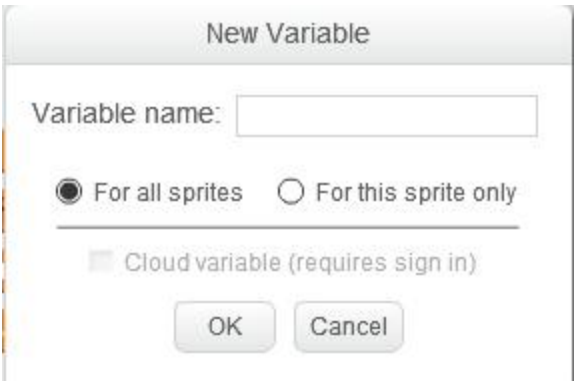


Image 2. Dialog for entering a variable name.

At that point, you simply type in the name for the new variable and click the **OK** button.

The radio buttons on that dialog allow you to specify which sprites have access to the variable.

The disabled cloud variable button (shown in light gray in [Image 2](#)) is a new feature of Scratch 2.0. Apparently registered users who are signed in can check that button and save the variable on the MIT website or on a website controlled by MIT. I wasn't signed in, which is why it was disabled.

Allowable variable names and lengths

Most programming languages have very specific requirements regarding the allowable characters for variable names. I suspect that this is also true of Scratch, but I haven't found that specification anywhere. You will probably be okay as long as you stick with letters, numbers, the underscore character "_", and the hyphen or minus sign "-".

It appears experimentally that the allowable length of the variable name is longer than you would ever want to use so length doesn't seem to be a limitation.

Many programming languages won't allow you to use a numeric character for the first character in a variable name, but that restriction doesn't seem to apply to Scratch. However, since you might later move up to more mainstream languages, you might want to avoid getting into the habit of using numeric characters as the first character in variable names.

Meaningful variable names

When using Scratch and all other programming languages, you should strive to use variable names that are meaningful. My preference is to begin variable names with a lower-case letter, use multiple words in the name where appropriate, and separate the words using a format commonly called *camelCase* to cause the human eye to separate the words.

(The upper-case characters are analogous to the humps on a camel.)

Here is an example of the camelCase format:

aVariableName

Some people prefer the following format:

A_Variable_Name

I prefer the *camelCase* format because it is easier to type, requires fewer characters, takes less space, and in my opinion, is just as effective.

Writing the program

Generally speaking, programs are written in Scratch by:

1. Selecting buttons in the upper-center of [Image 1](#) to expose the programming blocks in the tool boxes associated with each of the categories listed [earlier](#).
2. Dragging programming blocks from the toolbox in [Image 1](#) to the rightmost pane in [Image 1](#) and snapping those blocks together in groups to form program scripts.
3. Entering literal numeric values or **dragging other blocks** and dropping them into text fields with the same shape to fill out the details of the program.

There are other steps involved in writing a complex program that I didn't include above, but we will get to them later in this collection of modules. Hopefully you already know the physical steps in writing a program as a result of working through some online tutorials.

Although Scratch 2.0 has some new features and a different screen layout, the general procedure for writing a program in Scratch 2.0 is essentially the same as the procedure for writing a program in Scratch 1.4. Thus, the older v1.4 tutorials are still very relevant.

As you can see in [Image 1](#), this program consists of three scripts in the rightmost pane. Each script contains one tan block and two orange blocks. We already know that orange blocks have to do with data or variables. As you can see from the colors on the left ends of the buttons in [Image 1](#), tan blocks are related to **Events**.

Programming blocks in the Events category

[Image 3](#) shows some of the programming blocks that are available in the **Events** category. (Scratch 1.4 had tools for the **Events** category and the **Control** category combined into a single toolbox.)

Image 3. Programming blocks in the Events category.

Figure



Image 3.
Programming blocks
in the Events
category.

The finished program

The rightmost panel in [Image 1](#) shows the finished program. I created the program by dragging blocks from the **Data** toolbox and the **Events** toolbox and snapping them together to form three separate scripts.

(Actually the program that you see in [Image 1](#) is the result of an automatic conversion from v1.4 to v2.0 that was performed by the folks at MIT during the transition period from v1.4 to v2.0. I originally wrote that program in v1.4. Some of the toolboxes had different names in v1.4.)

Note that I also entered literal values of 0, 1, 5, -1, and -5 into the white text fields in the programming blocks for the variables after dragging them into the programming area.

In addition, I used the pull-down list on each of six orange **Data** blocks in [Image 1](#) to select the name of the variable to which that block applies. (Note that each of those six blocks refers either to **Counter** or **counter** .)

Behavior of the program

Click the green flag

The program contains three scripts. Each script remains silent until a specific event occurs. As the label on the uppermost block in the top programming script in [Image 1](#) indicates, the code in the top script in [Image 1](#) is executed once each time the user clicks the green flag shown in the upper right of the stage. This code causes the values stored in each of the two variables to be set to zero. This, in turn causes the variable displays in the Stage area to each show a value of 0.

Press the space bar

Because of the top block in the middle script reads "when space key pressed" , the code in the middle script is executed once each time the user presses the space bar.

The pull-down list on this block allows you to select among the keys on the keyboard with the space bar being the default.

One of the orange programming blocks in the middle script causes the value of the variable named **Counter** to change by +1 when the space bar is pressed. The other orange programming block causes the value of the variable named **counter** to change by +5 when the space bar is pressed.

In other words, repetitively pressing the space bar causes the **Counter** variable to count up in increments of one and causes the **counter** variable to count up in increments of five.

Click the mouse on the Stage

Because of the top block in the bottom script reads "when this sprite clicked" , the code in the bottom script in [Image 1](#) is executed once each time the user clicks the mouse in the large white Stage area shown in [Image 1](#).

(In this case, there are no sprites so this event is triggered by clicking the stage. I believe this is a new label on this block in v2.0. This label, which refers to a sprite, seems to be a little too specific.)

The bottom two blocks in this script are the same as the bottom two blocks in the middle script except that the algebraic signs on the two literal values are minus instead of plus.

The absence of a "-" character causes a literal value to be positive. A "+" character is not required for positive literal values.

Repetitively clicking the mouse in the Stage area causes the **Counter** variable to count down in increments of one and causes the **counter** variable to count down in increments of five.

Event-driven programming

The ability to write programs that cause certain operations to occur in response to events (*such as pressing the space bar and clicking the mouse on the Stage*) is often referred to as *event-driven programming*.

It is easy to write event-driven programs in Scratch. On the other hand, writing event-driven programs is not so easy in languages such as Java, C#, and C++. You need quite a lot of programming knowledge to write event-driven programs in those languages.

It is useful to give programming students a taste of event-driven programming early, if for no other reason than the fact that it tends to make programming more interesting.

Example blocks with pointed ends

The blocks shown in [Image 4](#) are not used in this program. They are shown in this module to illustrate blocks with pointed ends referred to [earlier](#).

Image 4. Example blocks with pointed ends.

Figure



Image 4. Example blocks with pointed ends.

Run the program

I encourage you to use the information provided above along with the Scratch 2.0 programming interface shown in [Image 1](#) to write and run this program. Experiment with the code, making changes, and observing the results of your changes. Make certain that you can explain why your changes behave as they do.

A copy of this program has been [posted online](#) for your review.

I also encourage you to write the programs described below.

Student programming projects

The following projects are designed to:

- Help the student retain the knowledge gained by studying the module.
- Require the student to think beyond the material presented in the module by requiring the student to answer the question "How do I ...?"

Project 1

Begin with the program named **Variable01** and modify it to create a new program named **Variable02** . Modify the original program in such a way that pressing the space bar five times in succession, (*after clicking the green flag to set both variables to zero*) , will cause the displayed values of the variables named **Counter** and **counter** to be as shown in [Image 5](#).

Image 5. Output from Project 1.

Figure

Space bar press	Counter value	counter value
1	1	1
2	2	3
3	3	6
4	4	10
5	5	15

Image 5. Output from Project 1.

Having reached that point, clicking the mouse five times in succession in the Stage area will cause the displayed values of the variables named **Counter** and **counter** to be as shown in [Image 6](#).

Image 6. Output from Project 1.

Figure

Mouse click	Counter value	counter value
1	4	19
2	3	22
3	2	24
4	1	25
5	0	25

Image 6. Output from Project 1.

A copy of this program is [posted online](#) for your review.

Project 2

You should successfully complete [Project 1](#) before attempting this project. Begin with the program named **Variable01** and modify it to create a new

program named **Variable03** . Modify the original program in such a way that pressing the space bar five times in succession, (*after clicking the green flag to set both variables to zero*) , will cause the displayed values of the variables named **Counter** and **counter** to be as shown in [Image 7](#).

Image 7. Output from Project 2.

Figure

Space bar press	Counter value	counter value
1	1	2
2	2	6
3	3	12
4	4	20
5	5	30

Image 7. Output from Project 2.

Having reached that point, clicking the mouse five times in succession in the Stage area will cause the displayed values of the variables named **Counter** and **counter** to be as shown in [Image 8](#).

Image 8. Output from Project 2.

Figure

Mouse click	Counter value	counter value
1	4	38
2	3	44
3	2	48
4	1	50
5	0	50

Image 8. Output from Project 2.

Hint: Compare the values of the variable named **counter** in [Project 2](#) with the values of the variable named **counter** in [Project 1](#) to determine what

you need to do to successfully write [Project 2](#).

A copy of this program is [posted online](#) for your review.

Summary

I began by providing an overview of computer programming. Then I provided technical explanations for memory, variables, and literals.

I explained the meaning of the term algorithm.

I presented and explained a Scratch program that is designed to illustrate the creation and use of variables in a computer program. The Scratch program also illustrates case sensitivity in variable names along with the ability to detect and respond to events fired by the keyboard and events fired by the mouse.

Finally I presented two student programming projects that are designed to:

- Help the student retain the knowledge gained by studying the module.
- Require the student to think beyond the material presented in the module by requiring the student to answer the question "How do I ...?"

What's next?

In the next module, I will continue the process of helping you to learn about the following computer programming concepts using Scratch 2.0:

- Variables
- Expressions and Operators
- Sequence, Selection, and Loop Structures

Resources

- [Scratch home](#)
- [Scratch download page](#)
- [Scratch tutorial - Dance Tutorial](#)

- [Scratch forums](#)
- [Son of String Art](#)
- [Scratch explanatory video](#)
- [Scratch tutorials from MIT](#)
- [Scratch tutorial - Digital Art for All](#)
- [Scratch for Budding Computer Scientists](#)
- [Learn Scratch at learnscratch.org](#)
- [Scratch Tutorial - Space Shuttle Mission STS-2020](#)
- [SCRATCH TUTORIALS ICT In Primary Education 2012-2013](#)
- [Scratch Wiki](#)
- [Scratch WikiTable of ContentsWebsite - Scratch Wiki](#)
- [Scratch WikiTable of ContentsTutorials - Scratch Wiki](#)
- [Scratch WikiTable of ContentsProgram - Scratch Wiki](#)
- [Tutorial about Variables - Scratch Wiki](#)
- [Scratch support MIT web site](#)
- [Scratch resources at Scratch.ie](#)
- [ScratchEd Resources](#)
- [Scratch project Day Dream](#)
- [Scratch project Son of String Art](#)
- [Scratch Project Scratch Tutorial](#)
- Online version of program named [Variable01](#)
- Online version of student programming project named [Variable02](#)
- Online version of student programming project named [Variable03](#)

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Scr0320: Memory, Variables, and Literals in Scratch 2.0
- File: Scr0320.htm
- Published: 05/11/13

Note: Disclaimers:

Financial : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.

I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module. In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.

Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Scr0330: Sequence, Selection, and Loop in Scratch 2.0.

The purpose of this module is to teach you about structured programming; the sequence structure, the selection structure, and the loop structure. The module will also teach you how to write a Scratch program that illustrates the selection structure, mouse events, and Cartesian coordinates.

Table of Contents

- [Preface](#)
 - [Viewing tip](#)
 - [Images](#)
- [General background information](#)
 - [What is structured programming?](#)
 - [One door in and one door out](#)
 - [Nesting of structures is allowed](#)
 - [Pseudocode](#)
 - [The sequence structure](#)
 - [The action elements themselves may be structures](#)
 - [The selection structure](#)
 - [Test a condition for true or false](#)
 - [The action elements themselves may be structures](#)
 - [Sometimes no action is required on false](#)
 - [The loop structure](#)
 - [Perform the test and exit on false](#)
 - [Perform some actions and repeat the test on true](#)
 - [Each action element may be another structure](#)
 - [Need to avoid infinite loops](#)

- [Other possible structures](#)
- [Preview](#)
 - [Programming interface for the program named IfSimple01](#)
 - [Click the green flag or the basketball](#)
- [Discussion and sample code](#)
 - [Three sprites](#)
 - [Program code for the LeftBeachball](#)
 - [Adding the blue go to block to the program](#)
 - [Coordinate values](#)
 - [Behavior of the LeftBeachball](#)
 - [Positioning the other beachball](#)
 - [The basketball](#)
 - [Initializing the position of the basketball](#)
 - [Initializing the orientation of the basketball](#)
 - [Handling mouse events on the basketball](#)
 - [Behavior of the bottom script](#)
 - [The selection block](#)
 - [Two independent decisions](#)
 - [Control structures available in Scratch](#)
 - [How do the two *if* blocks differ?](#)
 - [Specifying the condition on which the decision will be based](#)
 - [Two groups of programming blocks have the correct shape](#)
 - [Will use the touching block](#)
 - [Go back and examine the script](#)
 - [Conditions have been established - need actions](#)
 - [Programming blocks belonging to the Motion group](#)

- [Turn around and face the other way](#)
 - [An online version of this program is available](#)
- [Run the program](#)
- [Student programming project](#)
 - [Orthogonal axes in Cartesian coordinates](#)
 - [Draw a straight line to the location of the next mouse click](#)
 - [A sneak peek at the solution](#)
- [Summary](#)
- [What's next?](#)
- [Resources](#)
- [Miscellaneous](#)

Preface

[Scratch 2.0](#) (*released May 9, 2013*) is the second major version of Scratch to be released during the life of the product. Among other things, it features a redesigned editor and website, and allows you to edit projects directly from your web browser.

This module is one in a collection of modules designed to help beginners of all ages (*8 and up*) learn how to create the code for computer programs using [Scratch 2.0](#). Information is provided not only for the beginners themselves but also for their parents and teachers where appropriate.

The purpose of this module is to teach you about structured programming; the sequence structure, the selection structure, and the loop structure. The module will also teach you how to write a Scratch program that illustrates the selection structure, mouse events, and [Cartesian coordinates](#).

Finally, the module presents a student project by which you can demonstrate your understanding of the concepts learned from studying the module.

Viewing tip

I recommend that you open another copy of this module in a separate browser window and use the following links to easily find and view the images while you are reading about them.

Images

- [Image 1](#). The sequence structure in pseudocode.
- [Image 2](#). The selection structure in pseudocode.
- [Image 3](#). The loop structure in pseudocode.
- [Image 4](#). Scratch 2.0 programming interface for the program named IfSimple01.
- [Image 5](#). A paraphrased version of the Scratch code.
- [Image 6](#). Program code for the left beachball
- [Image 7](#). Initializing the position and orientation of the basketball.
- [Image 8](#). All of the code that applies to the basketball.
- [Image 9](#). Pseudocode for a selection structure.
- [Image 10](#). Two independent decisions.
- [Image 11](#). Control structures available in Scratch.
- [Image 12](#). Programming blocks belonging to the Sensing group.
- [Image 13](#). Initial output from the program named IfWithVar01.
- [Image 14](#). Program output after having clicked twice in the Stage area.

General background information

In this module, I will help you learn about:

- Structured programming.
- The *sequence* structure.
- The *selection* structure.
- The *loop* structure.

In introductory programming courses, you will often hear a lot about something called *structured programming*. In comparison with more modern and complex programming concepts such as *runtime*

polymorphism, structured programming is fairly mundane. However, that's not to say that structured programming isn't important. It is very important. But it is just a small bump in the road of learning that leads to a more complete understanding of computer programming, especially object-oriented programming.

What is structured programming ?

Basically, the concept of structured programming says that any programming logic problem can be solved using an appropriate combination of only three programming structures, none of which are complicated. The three structures are known generally as:

- The sequence structure.
- The **selection or decision structure** .
- The loop, repetition, or iteration structure.

One door in and one door out

To understand structured programming, you need to think in terms of a section of program code that has only one entry point and one exit point. It is very important that there cannot be multiple entry points or multiple exit points.

There must be only one way into the section of code and one way out of the section of code.

Nesting of structures is allowed

Another important part of the concept is that structures may be nested inside of other structures provided that every structure meets the basic rules for a structure.

Thus, by nesting simple structures inside of simple structures, large and complex overall structures can be constructed.

Pseudocode

Computer programming source code consists generally (*but not in Scratch*) of programming instructions written in text form with a very specific format or syntax that is designed to be understood by a computer program. (*This is commonly called source code.*) Humans who are not computer programmers might not be expected to understand much of what they see in computer programming source code.

According to [Wikipedia](#),

"The prefix **pseudo** - (from Greek... "lying, false") is used to mark something as false, fraudulent, or pretending to be something it is not."

Also according to [Wikipedia](#),

"Pseudocode is an informal high-level description of the operating principle of a computer program or other algorithm. It uses the structural conventions of a programming language, but is intended for human reading rather than machine reading. Pseudocode typically omits details that are not essential for human understanding of the algorithm..."

The purpose of using pseudocode is that it is easier for people to understand than conventional programming language code, and

that it is an efficient and environment-independent description of the key principles of an algorithm."

The *sequence* structure

We can describe the *sequence* structure using the pseudocode shown in [Image 1](#).

Image 1. The sequence structure in pseudocode.

Figure

```
Enter
  Perform one or more actions in sequence
Exit
```

Image 1. The sequence structure in pseudocode.

Thus, the general requirement for the sequence structure is that one or more actions may be performed in sequence after entry and before exit.

There may not be any branches or loops between the entry and the exit.

All actions must be taken in sequence.

The action elements themselves may be structures

However, it is important to note that one or more of the action elements may themselves be sequence, selection, or loop structures.

If each of the structures that make up the sequence has only one entry point and one exit point, each such structure can be viewed as a single action element in a sequence of actions.

The sequence structure is the simplest of the three, and there's not much more that I can say about it.

The *selection* structure

The *selection* or *decision* structure can be described as shown in the pseudocode in [Image 2](#).

Image 2. The selection structure in pseudocode.

Figure

```
Enter
  Test a condition for true or false
  On true
    Take one or more actions in sequence
  On false
    Take none, one, or more actions in sequence
Exit
```

Image 2. The selection structure in pseudocode.

Test a condition for true or false

Once again, there is only one entry point and one exit point.

The first thing that happens following entry is that some condition is tested for *true* or *false* .

The concept of something being *true* or *false* is commonly referred to as a *boolean condition* in computer programming (*named after George Boole*) .

If the condition is true, one or more actions are taken in sequence and control exits the structure.

If the condition is false, **none** , one or more different actions are taken in sequence and control exits the structure. (*Note the inclusion of the word **none** here.*)

The action elements themselves may be structures

Once again, each of the action elements in the sequence may be another sequence, selection, or loop structure.

Eventually all of the actions for a chosen branch will be completed in sequence and control will exit the structure.

Sometimes no action is required on false

It is often the case that no action is required when the test returns false. In that case, control simply exits the structure without performing any actions.

The *loop* structure

The *loop* or *iteration* structure can be described as shown in the pseudocode in [Image 3](#).

Image 3. The loop structure in pseudocode.

Figure

```
Enter
  Test a condition for true or false
  Exit on false
  On true
    Perform one or more actions in sequence.
    Go back and test the condition again
```

Image 3. The loop structure in pseudocode.

As before, there is only one entry point and one exit point. Note that in this case, the exit point is not at the end of the pseudocode. Instead, it follows the test.

Perform the test and exit on false

The first thing that happens following entry is that a condition is tested for true or false.

If the test returns false, control simply exits the structure without taking any action at all.

Perform some actions and repeat the test on true

If the test returns true:

- One or more actions are performed in sequence.
- The condition is tested again and the process is repeated.

During each *iteration*, if the test returns false, control exits the structure. If the test returns true, the entire process is repeated.

Each action element may be another structure

Each of the action elements may be implemented by another sequence, selection, or loop structure.

Eventually all of the actions will be completed and the condition will be tested again.

Need to avoid infinite loops

Generally speaking, unless something is done in one of the actions to cause the test to eventually return false, control will never exit the loop.

In this case, the program will be caught in what is commonly called an *infinite loop*.

Other possible structures

In some programming languages, there are structures other than sequence, selection, and loop that structured-programming experts are willing to accept *for convenience* including:

- The switch-case structure.
- The do-until structure.
- The for loop
- The for-each loop

While sometimes more convenient than the three main structures, these structures are not required for the solution of programming logic problems.

Preview

In this module, I will present and explain the simplest example of a selection structure that I was able to write in Scratch without using variables and without using relational or logical operators. (*I will explain operators, including relational and logical operators in future modules.*)

Programming interface for the program named IfSimple01

The program places a basketball and two beachballs on the Stage as shown in [Image 4](#).

Image 4. Scratch 2.0 programming interface for the program named IfSimple01.

Figure

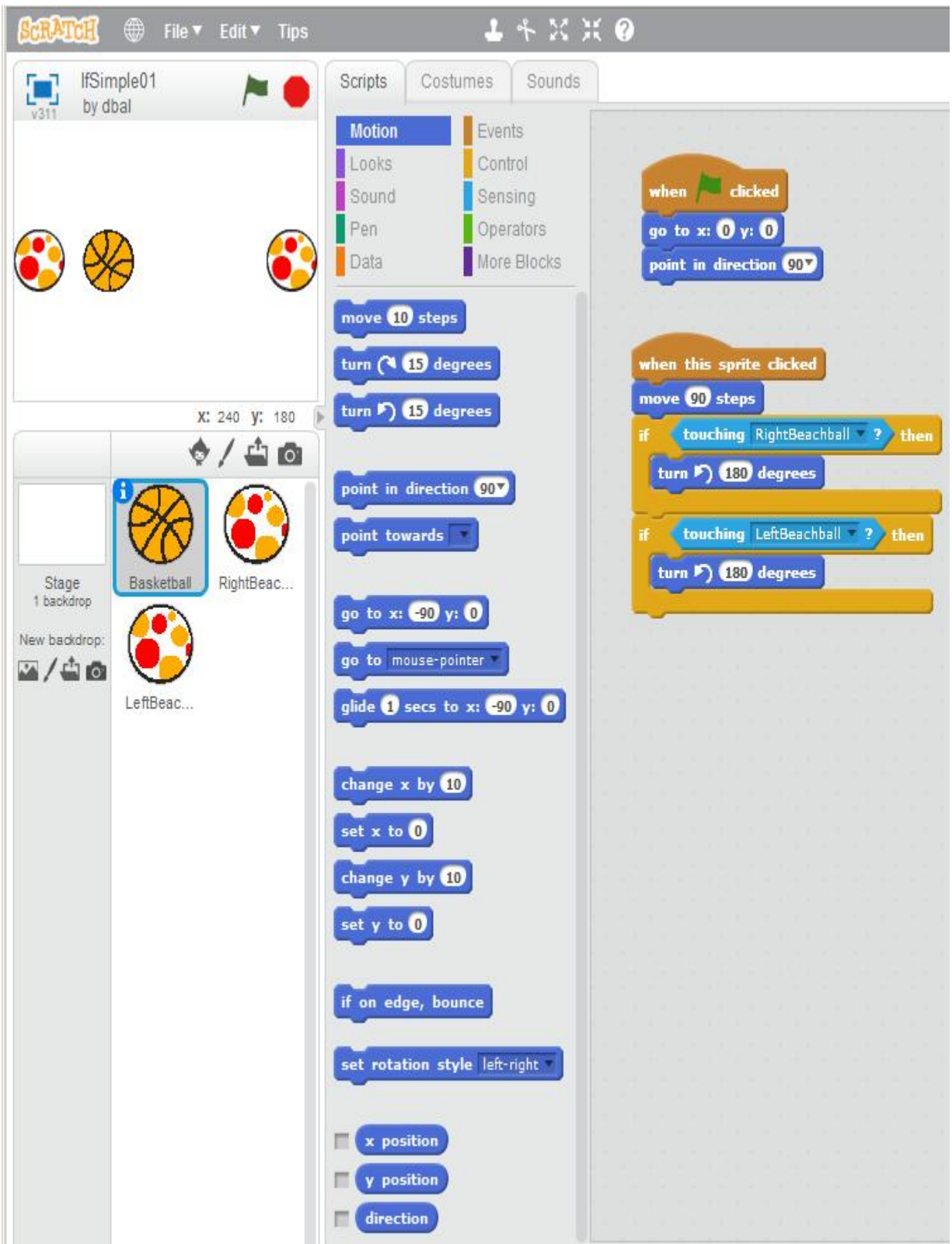


Image 4. Scratch 2.0 programming interface for the program named

IfSimple01.

Click the green flag or the basketball

When the user clicks the green flag in the upper right corner of the stage, the three balls are placed in a horizontal line with the basketball in the center.

Scratch code, which can be paraphrased as shown in [Image 5](#), is executed each time the user clicks the basketball with the mouse.

Image 5. A paraphrased version of the Scratch code.

Figure

```
when Basketball is clicked{
  move basketball forward by 90 steps
  if(Basketball is touching RightBeachball){
    turn Basketball by 180 degrees
  }//end if

  if(Basketball is touching LeftBeachball){
    turn Basketball by 180 degrees
  }//end if
}
```

Image 5. A paraphrased version of the Scratch code.

In other words, if you repetitively click the basketball with the mouse, it will move back and forth from left to right bouncing off of the two beachballs. The basketball will keep bouncing back and forth between the two beachballs for as long as you continue clicking on the basketball.

Discussion and sample code

Let's walk through the steps required to develop this program. I will deal first with the code that defines the behavior of the program when the user clicks the green flag in the upper right corner of the stage in [Image 4](#).

The Cartesian coordinate system The position of sprites on the Stage in the Scratch programming interface is based on a two-dimensional [Cartesian coordinate system](#) with the origin at the center of the Stage. The x coordinates range from -240 at the left to +240 at the right. The y coordinates range from +180 at the top to -180 at the bottom.

Three sprites

The area below the stage in [Image 4](#) shows that this program contains two beachballs and one basketball in addition to the Stage. (*The beachballs and the basketball are sprites.*)

You add a sprite to your program by clicking one of the four buttons immediately below the stage. If you hover your mouse over those buttons going from left to right, the following tooltips appear:

1. Choose a sprite from library
2. Paint new sprite
3. Upload sprite from file
4. New sprite from camera

In this program, the beachballs and the basketball were added by selecting the first or leftmost button in the row of four buttons and then selecting the appropriate sprite from the library.

Although not visible in the cropped version of the programming interface in [Image 4](#), when you click on one of the sprites that you have added to the

program, that sprite appears at the top of the rightmost pane. (See [Image 6](#) for example.) Having done that, you can then drag programming blocks into the rightmost pane that define the behavior of that sprite.

Program code for the LeftBeachball

The code that I wrote (by dragging blocks into the rightmost pane in the programming interface) for the sprite named **LeftBeachball** is shown in [Image 6](#). (Note the image of the beachball in the upper right corner. When this is an image of a beachball, all of the scripts in the rightmost panel apply to that particular beachball sprite.)

Image 6. Program code for the left beachball.

Figure



Image 6. Program code for the left beachball.

You are already familiar with the tan block with the green flag shown in [Image 6](#) because you learned about it in a previous module. However, the blue block in [Image 6](#) has not been used prior to this module.

Adding the blue *go to* block to the program

The blue block shown in [Image 6](#) was added to the program module by:

- Clicking the dark blue button labeled **Motion** in the upper center of [Image 4](#).
- Dragging the blue block shown in [Image 6](#) from the toolbox to the rightmost panel and clicking it into place under the tan block.
- Typing the literal values -200 and 0 into the two white boxes on the blue block.

Coordinate values

If you move the mouse pointer around the stage in the programming interface, the coordinates of the mouse pointer are displayed immediately below the stage as shown in [Image 4](#). As mentioned earlier, the x coordinates range from -240 at the left to +240 at the right. The y coordinates range from +180 at the top to -180 at the bottom.

Behavior of the LeftBeachball

The behavior of the script shown in [Image 6](#) can be interpreted as follows: When the user clicks the green flag, cause the sprite named **LeftBeachball** to move to a location with an x (*horizontal*) coordinate value of -200 and a y (*vertical*) coordinate value of 0. Since the origin is at the center of the Stage, this causes the beachball to move to the left of the origin on the horizontal axis.

Some experimentation was required

Because I didn't know the diameter of the beachball, I had to experiment to determine how far to move it to the left of the origin to locate it at the left side of the Stage as shown in [Image 4](#). I settled on a value of -200 for the x coordinate and a value of 0 for the y coordinate.

Positioning the other beachball

In the interest of brevity, I won't show the code required to position the beachball on the right side of the Stage. I did exactly the same thing for that beachball except that I specified the value of the x coordinate to be 200 instead of - 200. This causes the beachball named **RightBeachball** to move to the right side of the Stage when the user clicks the green flag.

The basketball

Initializing the position of the basketball

[Image 7](#) shows a portion of the rightmost panel after clicking on the sprite named **Basketball** . *(Note the image of the basketball in the upper right corner. When this is an image of the basketball, all of the scripts in the rightmost panel apply to the basketball sprite.)*

Image 7. Initializing the position and orientation of the basketball.

Figure

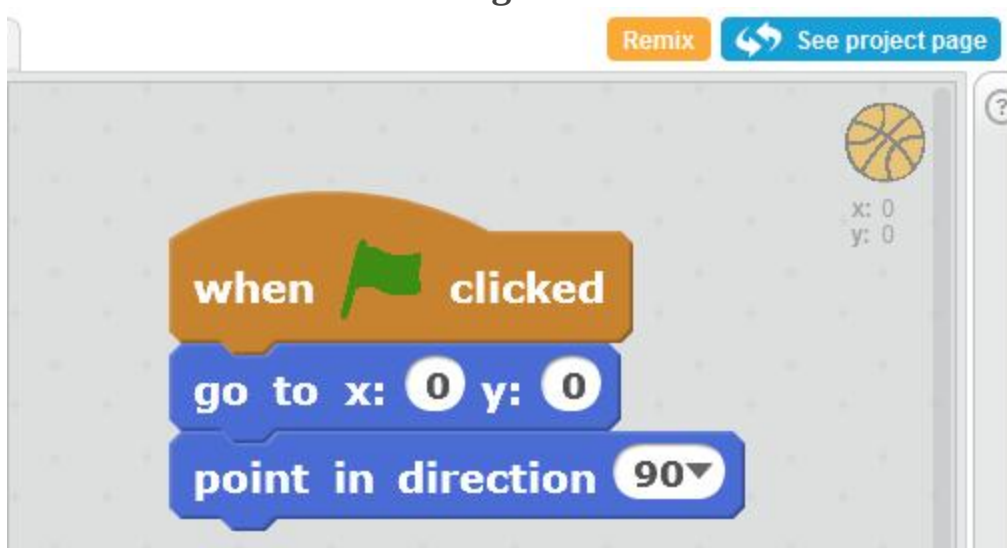


Image 7. Initializing the position and orientation of the basketball.

You will note that the tan block and the uppermost blue block in [Image 7](#) are the same as in [Image 6](#) except that the x coordinate value is set to 0. This causes the basketball to move to the origin when the user clicks the green flag.

Initializing the orientation of the basketball

However, [Image 7](#) contains a block that is not contained in [Image 6](#). The bottom blue block in [Image 7](#) is used to set the *orientation* of the basketball. (By orientation, I mean the direction that the basketball is facing.)

It may seem strange to say that a round basketball is facing in one direction or the other. However, every sprite has a front, back, top, and bottom even in those cases where it is not visually obvious. The orientation would be visually obvious if I were to use an animal for the sprite in place of the basketball. You can tell which direction the basketball is facing by observing the curved diagonal lines on the basketball.

After you drag the blue block labeled **point in direction** into the rightmost panel, you can click the arrow in the white box to expose the following four choices:

- (90) right
- (-90) left
- (0) up
- (180) down

As you can see, I selected the choice that causes the basketball to face to the right. As a result, when the user clicks the green flag, the basketball will move to the origin and turn to face the right.

Handling mouse events on the basketball

[Image 8](#) shows all of the code in the rightmost panel that applies to the basketball.

Image 8. All of the code that applies to the basketball.

Figure

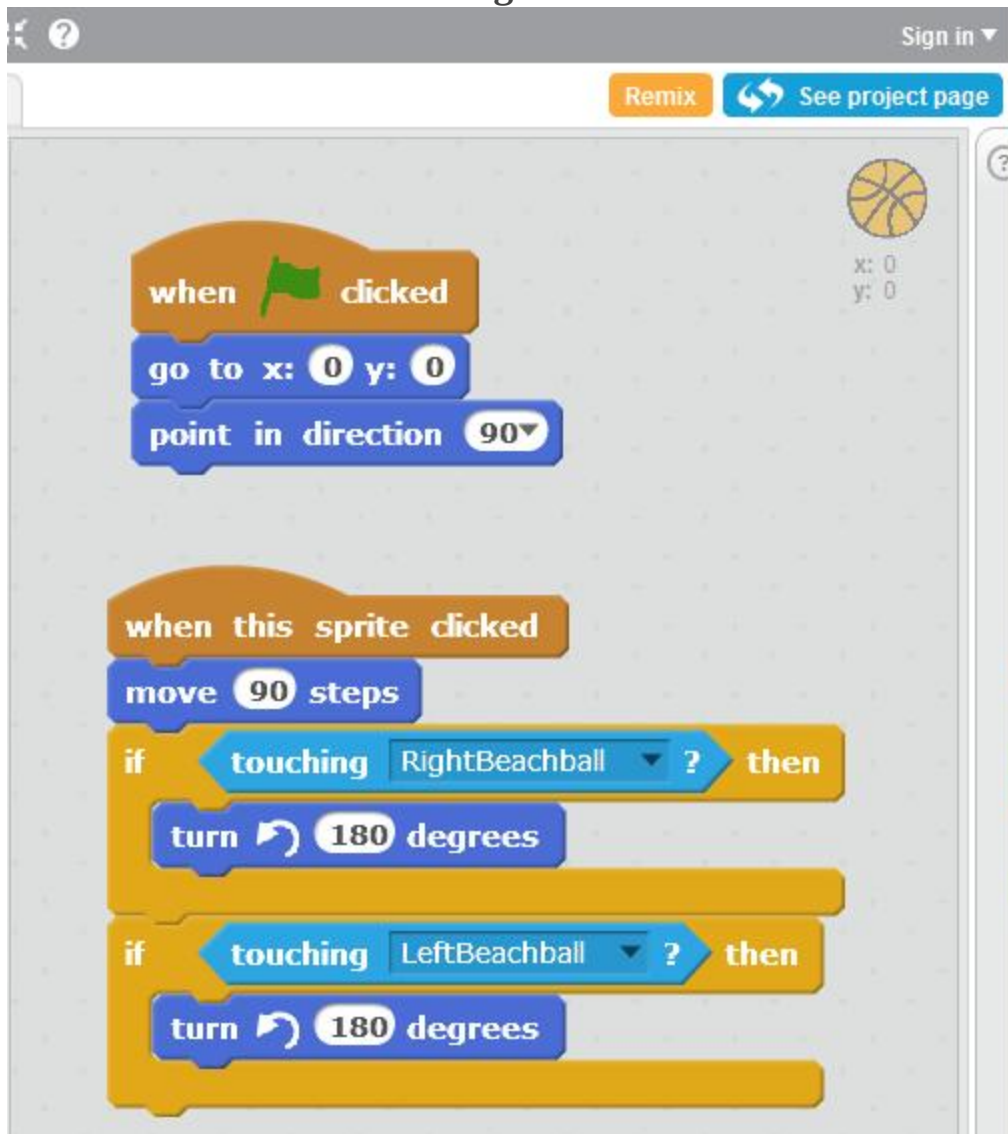


Image 8. All of the code that applies to the basketball.

The top script in [Image 8](#) is another copy of the script that was shown in [Image 7](#). At this point, we are interested in the behavior of the bottom

script in [Image 8](#).

Behavior of the bottom script

The dark tan block labeled **when this sprite clicked** specifies that all of the actions produced by the blocks attached to that block will occur when the user clicks the basketball with the mouse. Furthermore, those actions will occur in top-to-bottom order.

The blue block labeled **move 90 steps** causes the basketball to move 90 steps forward (*in the direction that it is facing*). Each step constitutes one unit in the Cartesian coordinate system. For example, if you were to change this value from 90 to 240, that would cause the basketball to move from the origin to the extreme right edge of the Stage.

The selection block

As I indicated [earlier](#), the *selection* structure is sometimes referred to as a *decision* structure. In other words, this structure causes the program to select or make a decision between two alternatives. The pseudocode in [Image 9](#) describes this process.

Image 9. Pseudocode for a selection structure.

Figure

```
if a specified condition is true
  perform a specified action
otherwise (when the specified condition is not
true)
  perform a different action
```

Image 9. Pseudocode for a selection structure.

Sometimes the second part of the selection process isn't needed. In other words, in some cases, if the specified condition is not true, there is no requirement to do anything at all. That is the case in this program.

Two independent decisions

This program makes two independent decisions shown by the pseudocode in [Image 10](#).

Image 10. Two independent decisions.

Figure

```
if the basketball is touching the beachball on the
right
    turn around and face to the left

if the basketball is touching the beachball on the
left
    turn around and face to the right
```

Image 10. Two independent decisions.

These two decisions are implemented by the two light tan blocks in [Image 8](#) containing the word **if**. (*From this point forward, I won't distinguish between dark tan and light tan.*)

*Clicking the tan **Control** button shown at the upper center in [Image 4](#) exposes a large number of programming blocks. Two of them are used to create pure selection structures (if-then and if-then-else) and one is used to create a combination of a loop structure and a selection structure (repeat-until).*

Control structures available in Scratch

Clicking the tan **Control** button shown at the upper center in [Image 4](#) exposes the blocks shown in [Image 11](#). As mentioned above, two of the blocks can be used to create pure selection structures and one can be used to create a combination of a loop structure and a selection structure. The other blocks serve other purposes.

Selection structures are created by dragging the selection blocks into the rightmost panel in order to apply them to a particular sprite.

Image 11. Control structures available in Scratch.

Figure



Image 11. Control structures available in Scratch.

How do the two *if* blocks differ?

The *if-then-else* block in [Image 11](#) is used to select between two specific actions (*or two sequences of actions*) . The *if-then* block is used when there is only one action (*or one sequence of actions*) and you need to decide whether to take that action or not.

That latter case describes the situation in this program. If the basketball is touching a beachball, a specific action is required. If the basketball is not touching a beachball, no specific action is required.

Specifying the condition on which the decision will be based

Note the empty darker depressed area in some of the blocks shown in [Image 11](#) (*the rectangles with the pointed ends*) . In order to use these blocks, you must drop another block having the same shape into the depressed area. The block that you drop into that area must specify the condition that will be used to make the decision.

Two groups of programming blocks have the correct shape

Unless I missed seeing some others, there are only two buttons at the top center of [Image 4](#) that expose blocks having the required shape:

- Sensing (light blue)
- Operators (green)

In this case, we will select and use a block from the light blue **Sensing** group. (*We will use programming blocks from the green **Operators** group in a future module.*) All of the blocks belonging to the **Sensing** group are shown in [Image 12](#).

Image 12. Programming blocks belonging to the Sensing group.

Figure



Image 12.
Programming blocks

belonging to the
Sensing group.

Will use the *touching* block

We will use the block labeled **touching** followed by a pull-down list and a question mark.

The items that appear in the pull-down list depend on the sprites that have been added to the program. For this program, that list consists of the following choices when the basketball has been selected for programming:

- mouse-pointer
- edge
- LeftBeachball
- RightBeachball

The top two choices are always there. The remaining choices depend on the sprites that have been added to the program and the sprite that has been selected for programming.

*(Note that the **Basketball** sprite does not appear in the above list because it was selected for programming when I examined the list. In other words, you can't determine that the basketball is touching itself.)*

Go back and examine the script

Going back to the bottom script in [Image 8](#), you can see that I dragged two copies of the **if** block shown in [Image 11](#) into the rightmost panel and connected the blocks as shown in [Image 8](#). Then I dragged two copies of

the **touching** block from the **Sensing** group shown in [Image 12](#), and dropped each of those blocks into the corresponding locations in the **if** blocks in [Image 8](#).

Then I selected **RightBeachball** from the pull-down list for one of the **touching** blocks and selected **LeftBeachball** from the pull-down list for the other **touching** block.

Conditions have been established - need actions

At this point, I had the conditions for the two selection structures established, but I hadn't yet specified the actions to be taken when one or the other of the conditions is found to be true.

Programming blocks belonging to the Motion group

[Image 4](#) shows all of the programming blocks that are exposed by clicking the **Motion** button at the top center of [Image 4](#).

You saw three of the programming blocks from [Image 4](#) being used in [Image 6](#), [Image 7](#), and immediately below the top tan block in the bottom script in [Image 8](#). Now we need to use another of the programming blocks from [Image 4](#).

Turn around and face the other way

Recall that the blue **point in direction** block in the top script in [Image 8](#) causes the basketball to turn to face to the right when the user clicks the green flag.

The two **turn** blocks in the bottom script in [Image 8](#) cause the basketball to rotate around its center by 180 degrees. This, in turn, causes it to face in the opposite direction from the direction that it was previously facing. *(It also turns it upside down, but that doesn't matter for a round basketball.)*

This is the action that is required whenever either of the **touching** blocks is true. In other words, whenever the basketball touches either of the beachballs, it must turn to face the opposite direction and be prepared to move 90 steps in that direction the next time the user clicks the basketball.

An online version of this program is available

A copy of this program has been posted online for your review (see [Resources](#) for the URL) . If you don't find the program using that URL, search the Scratch site for the user named dbal.

Run the program

I encourage you to use the information provided above to write this program. Experiment with the code, making changes, and observing the results of your changes. Make certain that you can explain why your changes behave as they do.

Just for fun, use blocks from the purple **Sound** group and add some sound effects to your program.

I also encourage you to write the program described below.

Student programming project

Write a Scratch program named **IfWithVar01** that produces the output shown in [Image 13](#) when the user clicks the green flag. (*Don't be concerned about matching the values in the x and y variables in [Image 13](#).*)

Image 13. Initial output from the program named IfWithVar01.

Figure

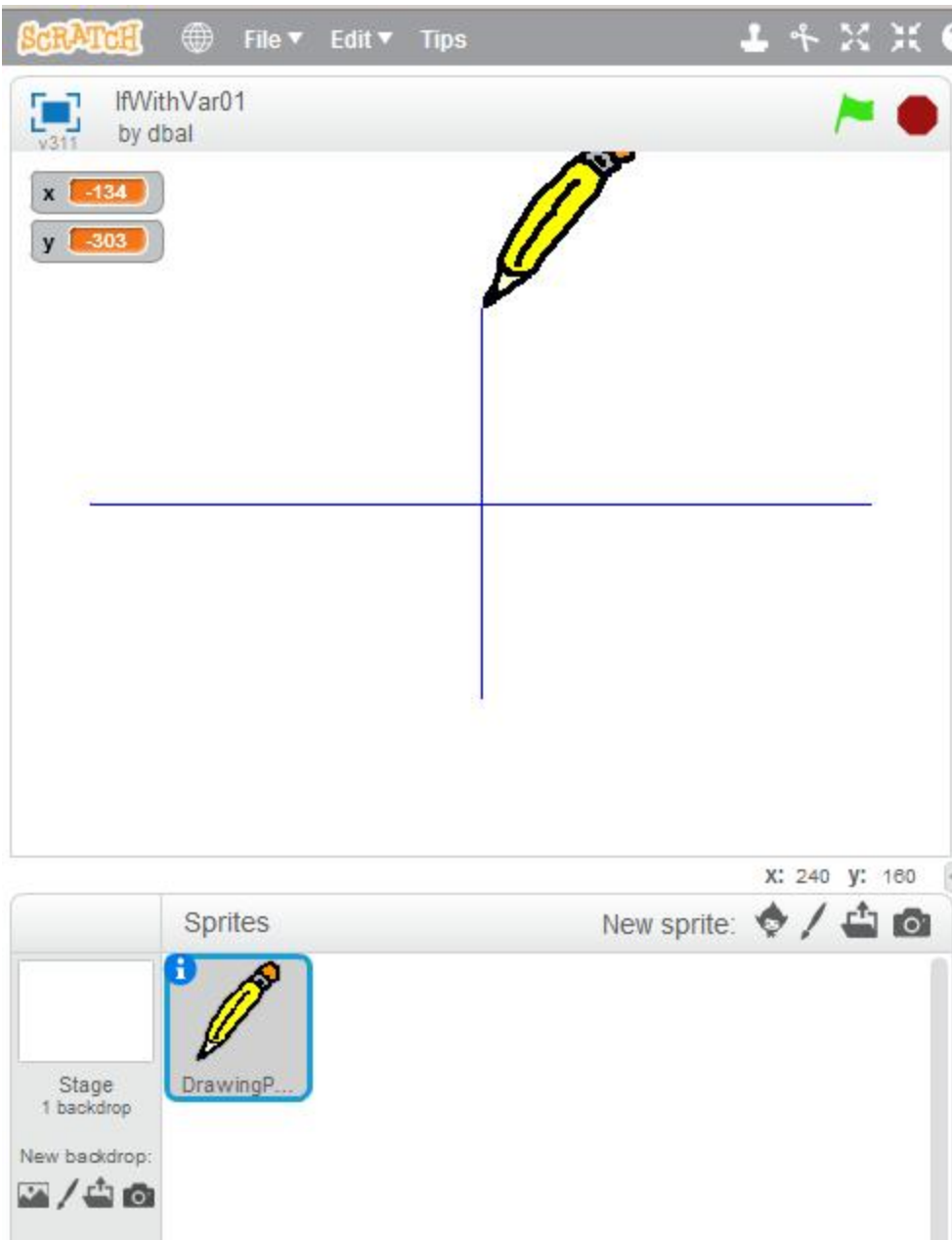


Image 13. Initial output from the program named IfWithVar01.

Orthogonal axes in Cartesian coordinates

When the user clicks the green flag, a **DrawingPencil** sprite draws a pair of orthogonal axes that intersect at the origin in the white Stage area. Make the horizontal axis extend from -200 to 200. Make the vertical axis extend from -100 to 100.

Draw a straight line to the location of the next mouse click

Each time the user clicks the mouse in the white Stage area (*after the user has clicked the green flag*), a straight line is drawn from the current location of the **DrawingPencil** to the location where the mouse click occurred. [Image 14](#) shows an example output after two mouse clicks.

Image 14. Program output after having clicked twice in the Stage area.

Figure

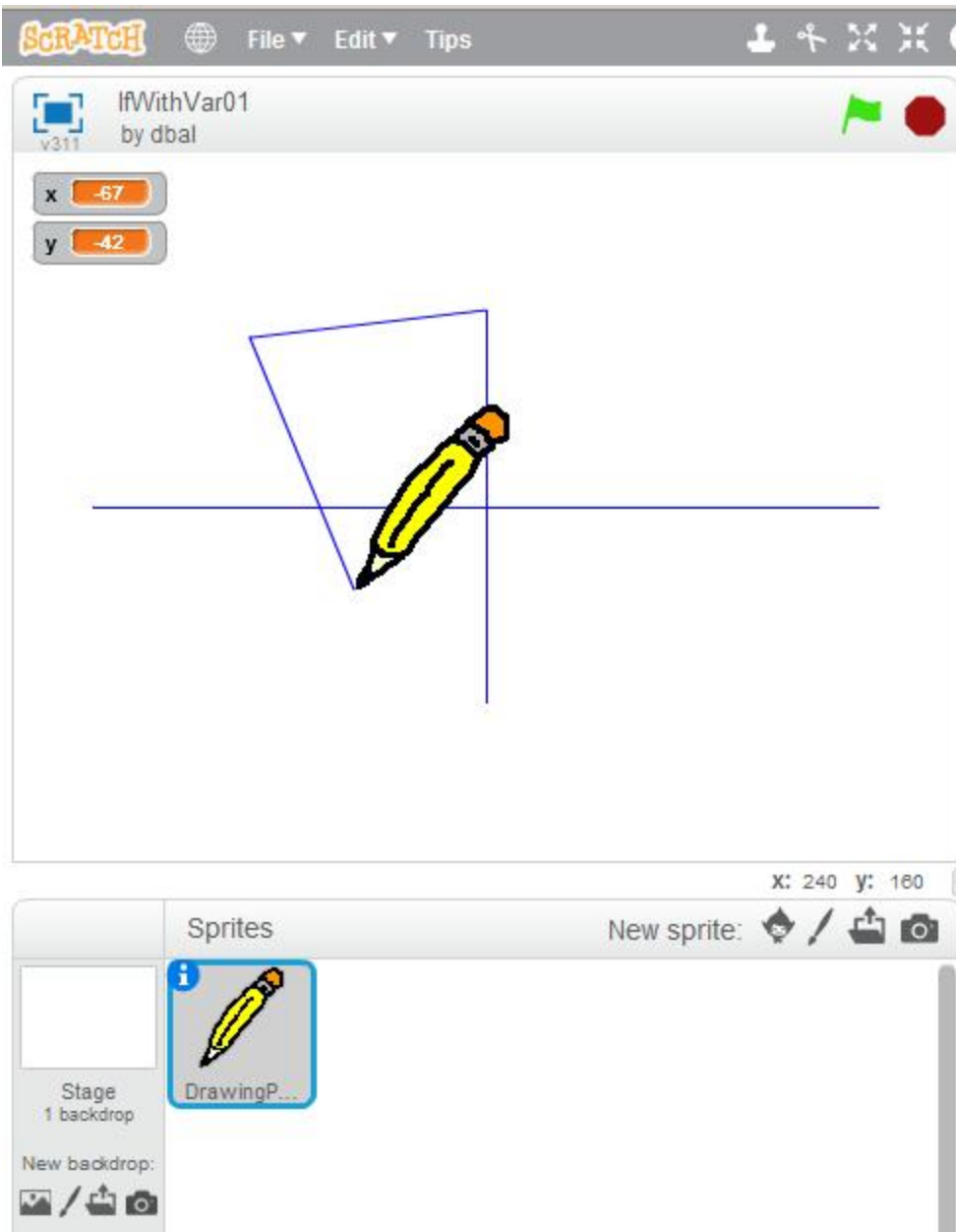


Image 14. Program output after having clicked twice in the Stage area.

A sneak peek at the solution

In case you need to sneak a peek at the solution to this programming project, a copy of this program has been posted online for your review (see [Resources](#) for the URL) . *(If you don't find the program using that URL, search the Scratch website for the user named **dbal** .)*

Once you locate the project on the Scratch web site, you can execute it online.

Summary

I began by explaining structured programming, the sequence structure, the selection structure, and the loop structure. Then I presented and explained a Scratch program that illustrates the selection structure. *(The sequence structure is so simple that it doesn't require an explanation. The loop structure will be explained in a future module.)* The program also illustrates the handling of mouse events and Cartesian coordinates.

Finally, I provided the specifications for a student-programming project for you to write to demonstrate your understanding of what you learned from the first program.

Copies of both programs have been posted online for your review (see [Resources](#) for the URL) . If you don't find the program using that URL, search the Scratch site for the user named **dbal** .

What's next?

In the next module, I will teach you about arithmetic operators. In the two modules following that one, I will teach you about relational and logical operators and how to use those operators to write the conditional expressions used in selection and loop structures.

Resources

- [Scratch home](#)
- [Scratch tutorials](#)

- [Scratch explanatory video](#)
- [Scratch tutorials from MIT](#)
- [Scratch tutorial - Digital Art for All](#)
- [Scratch for Budding Computer Scientists](#)
- [Learn Scratch at learnscratch.org](#)
- [Scratch Tutorial - Space Shuttle Mission STS-2020](#)
- [SCRATCH TUTORIALS ICT In Primary Education 2012-2013](#)
- [Scratch Wiki](#)
- [Scratch WikiTable of ContentsWebsite - Scratch Wiki](#)
- [Scratch WikiTable of ContentsTutorials - Scratch Wiki](#)
- [Scratch WikiTable of ContentsProgram - Scratch Wiki](#)
- [Tutorial about Variables - Scratch Wiki](#)
- [Scratch support MIT web site](#)
- [Scratch resources at Scratch.ie](#)
- [ScratchEd Resources](#)
- [Scratch project Day Dream](#)
- [Scratch project Son of String Art](#)
- [Scratch Project Scratch Tutorial](#)
- [Variable01](#) - Online version of program
- [Variable02](#) - Online version of student programming project
- [Variable03](#) - Online version of student programming project
- [IfSimple01](#) - Online version of program
- [IfWithVar01](#) - Online version of student programming project

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Scr0330: Sequence, Selection, and Loop in Scratch 2.0.
- File: Scr0330.htm
- Published: 05/11/13

Note: Disclaimers:

Financial : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.

I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module. In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.

Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Scr0340: Arithmetic Operators in Scratch 2.0

The purpose of this module is to teach you about operators and operands in general and arithmetic operators in particular. You will also learn about expressions and statements, and you will learn how to write a Scratch program that illustrates the use of arithmetic operators in Scratch.

Table of Contents

- [Preface](#)
 - [Viewing tip](#)
 - [Images](#)
- [General background information](#)
 - [Operators](#)
 - [Operands](#)
 - [Expressions](#)
 - [Statements](#)
 - [A brief word about type](#)
 - [Unary, binary, and ternary operators](#)
 - [Some operators can be either unary or binary.](#)
 - [The minus character as a unary operator](#)
 - [Binary operators use infix notation](#)
 - [General behavior of an operator](#)
 - [Operator categories](#)
- [Preview](#)
- [Discussion and sample code](#)
 - [Four variables](#)
 - [A button](#)
 - [Two scripts in the rightmost panel](#)
 - [Variables with sliders](#)

- [How to create a slider](#)
- [Expanded view of the rightmost panel](#)
- [Initialize the variable values to zero](#)
- [Define the behavior of the button](#)
 - [Constructing the script](#)
 - [Not exactly what we want to happen](#)
- [The Operators toolbox](#)
 - [Four arithmetic operators](#)
 - [Use an asterisk for multiplication](#)
 - [Use a forward slash for division](#)
 - [More blocks of the correct shape](#)
 - [Random numbers](#)
 - [The modulus](#)
 - [Various numeric representations](#)
 - [Rounding a number](#)
- [Drag and drop the plus and minus operators](#)
- [A few more steps are required](#)
- [Operation of the program](#)
- [An online version of this program is available](#)
- [Run the program](#)
- [Student programming project](#)
- [Summary](#)
- [What's next?](#)
- [Resources](#)
- [Miscellaneous](#)

Preface

[Scratch 2.0](#) (*released May 9, 2013*) is the second major version of Scratch to be released during the life of the product. Among other things, it features a redesigned editor and website, and allows you to edit projects directly from your web browser.

This module is one in a collection of modules designed to help beginners of all ages (*8 and up*) learn how to create the code for computer programs using [Scratch 2.0](#). Information is provided not only for the beginners themselves but also for their parents and teachers where appropriate.

The purpose of this module is to teach you about operators and operands in general and arithmetic operators in particular. You will also learn about expressions and statements, and you will learn how to write a [Scratch](#) program that illustrates the use of arithmetic operators in Scratch.

I will also provide the specifications for a student-programming project for you to complete in order to demonstrate your understanding of what you learned from the first program.

Viewing tip

I recommend that you open another copy of this module in a separate browser window and use the following links to easily find and view the images while you are reading about them.

Images

- [Image 1](#). Reduced screen shot of Arithmetic01 programming interface.
- [Image 2](#). Expanded view of the rightmost panel.
- [Image 3](#). Preparing to use the arithmetic operators.
- [Image 4](#). Result of dropping addition and subtraction operators into variable blocks.
- [Image 5](#). Stage area of the finished program.
- [Image 6](#). Making the button say ouch.
- [Image 7](#). Output from student project program named Arithmetic02.

General background information

Operators

Operators are the action elements of a computer program. They perform actions such as adding two variables, dividing one variable by another variable, comparing one variable to another variable, etc.

Operands

According to the current jargon, *operators* operate on *operands* .

For example, in the following expression, the plus character is an operator while **x** and **y** are operands.

x + y

Assuming that **x** and **y** are numeric variables, this expression produces the sum of the values stored in the variables named **x** and **y** . The variable **x** would be called the *left operand* and the variable **y** would be called the *right operand* .

Expressions

Computer programs in many languages consist of statements, which in turn, consist of expressions. (*Programming blocks substitute for statements in Scratch.*)

An expression is a specific combination of operators and operands that evaluates to a particular result. The operands can be variables, literals, or method calls. (*Note that Scratch v1.4 didn't support methods. However, Scratch 2.0 allows you to design and create your own blocks, which helps to alleviate that deficiency.*)

In your past experience, you may have referred to expressions by the names *formulas* or *equations* . Although formulas and equations are not exactly the same thing as expressions, they are close enough to help you understand what expressions are and how they are used.

Statements

A statement is a specific combination of expressions. The following is an example of a statement comprised of expressions in a text-based language such as Java, C++, or C#.

```
z = x + y;
```

Operationally, values are retrieved from the variables named **x** and **y** in the above statement. These two values are added together. The result is stored in (*assigned to*) the variable named **z** , replacing whatever value may previously have been contained in that variable.

The plus character (+) would commonly be called the addition operator or the concatenation operator, depending on the *type* of data stored in the variables named **x** and **y** . The equal character (=) would commonly be called the assignment operator in Java, Alice, C#, and C++, but we will see in a future module that it is also used as a *relational* operator in Scratch.

A brief word about type

As a very crude analogy, you can think of *variables* as the pens at the animal shelter and think of *type* as the kinds of animals that reside in those pens. Dogs and cats are different types of animals that don't usually coexist very well in the same pen. Therefore, they are normally put in different pens.

Similarly, different types of data don't coexist well in the same variable in *type-sensitive* languages. Therefore, in type-sensitive languages, there are

very stringent rules as to the types of data that can be stored in each variable.

Scratch has two types of data that can be stored in a variable (*numeric and string*) . However, Scratch is not a *type-sensitive* language so type is not an issue in Scratch. (*Scratch also has a semblance of a boolean type but it tends to take care of itself.*)

Unary, binary, and ternary operators

Many programming languages provide operators that can be used to perform an action on one, two, or three operands. I believe that operators in Scratch are confined to only one or two operands.

An operator that operates on one operand is called a **unary** operator. An operator that operates on two operands is called a **binary** operator. An operator that operates on three operands is called a **ternary** operator.

Some operators can be either unary or binary

Some operators can behave either as a unary or as a binary operator in languages such as Java and C#. The best-known operator that can behave either way in those languages is the minus character (-).

As a binary operator, the minus character causes its right operand to be subtracted from its left operand. For example, the third statement below subtracts the variable **y** from the variable **x** and assigns the result of the subtraction to the variable **z** . After the third statement is executed, the variable **z** contains the value 1.

- **x = 6;**
- **y = 5;**
- **z = x - y;**

The minus character as a unary operator

As a unary operator, the minus character causes the algebraic sign of the right operand to be changed. For example, the second statement below causes a value of -5 to be assigned to (*stored in*) the variable **x** .

- **y = 5;**
- **x = -y;**

Binary operators use infix notation

To keep you abreast of the current jargon, **binary** operators in Scratch always use *infix* notation. This means that the operator appears between its operands.

Some other programming languages have **unary** operators that use *prefix* notation and *postfix* notation. For prefix notation, the operator appears before (*to the left of*) its operand. For postfix notation, the operator appears after (*to the right of*) its operand.

General behavior of an operator

As a result of performing the specified action, an operator can be said to return a value (*or evaluate to a value*) of a given type. The type of value returned depends on the operator and the type of the operands.

To evaluate to a value means that after the action is performed, the operator and its operands are effectively replaced in the expression by the value that is returned.

Operator categories

There are many different kinds of operators. Therefore, the easiest way to study them is to divide them into categories such as the following

- arithmetic
- relational
- logical
- bitwise
- assignment

This module will concentrate on arithmetic operators. Future modules will deal with other kinds of operators.

Preview

In this module, I will present and explain a Scratch program named **Arithmetic01** . This program illustrates the use of the following arithmetic operators (*see the bottom script in the rightmost panel of [Image 4](#)*):

- + (addition)
- - (subtraction)

Variables with the following names are created and displayed on the stage (*see [Image 5](#)*):

- **LeftOperand** - has a slider
- **RightOperand** - has a slider
- **Sum**
- **Diff**

In addition, a button is displayed on the stage.

The user adjusts the values of the **LeftOperand** and **RightOperand** variables with a pair of sliders. When the user clicks the button, the variable named **Sum** displays the sum of the values of the left and right operands. The variable named **Diff** displays the result of subtracting the right operand from the left operand.

Discussion and sample code

I'm going to walk you through the steps required to develop this program, being brief on those things that you already know about and being more verbose on the new material.

To begin with, [Image 1](#) shows a reduced screen shot of the online programming interface in the browser window. Note that the source code in the rightmost panel was reduced in size to help it fit into this presentation format. The stage was also switched to the small format by clicking the little triangle immediately below the stage.

Image 1. Reduced screen shot of Arithmetic01 programming interface.

Figure

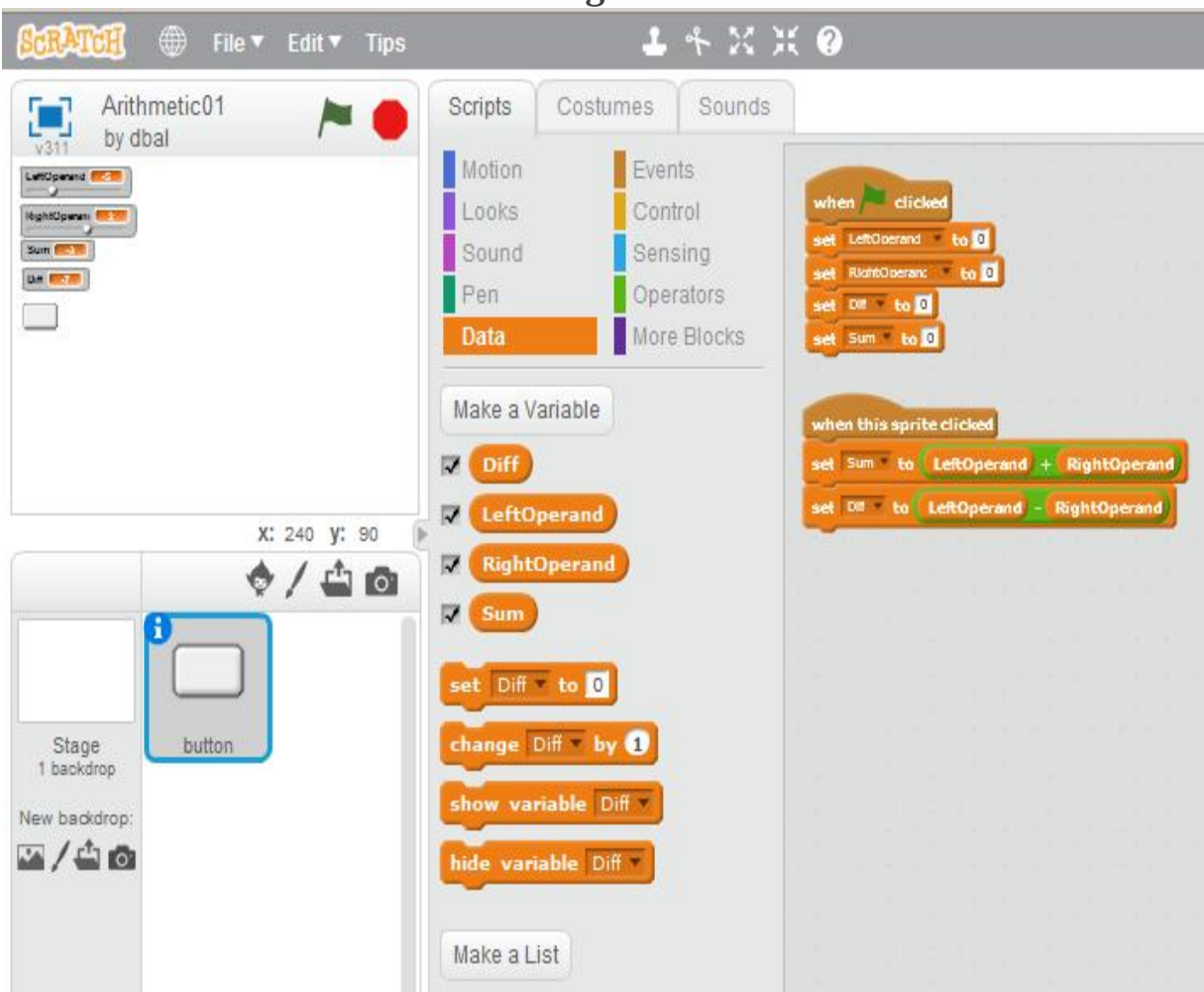


Image 1. Reduced screen shot of Arithmetic01 programming interface.

[Image 1](#) lacks detail due to the requirement to reduce the size of the image for publication in this format, but there are some aspects of [Image 1](#) that can be seen.

Four variables

The center panel (*and the stage*) shows that four variables have been created. In case you can't read the names of those variables, I will list them here:

- **LeftOperand**
- **RightOperand**
- **Sum**
- **Diff**

You can probably see that the checkbox next to each variable has been checked. As you learned in an earlier module, checking this box causes the variable to be displayed in the Stage area on the upper left. Thus you can see all four variables being displayed in the upper left corner of the Stage.

A button

Skipping to the area immediately below the Stage, you can see that a button sprite has been added to the program. That button has been selected in [Image 1](#), making it possible to drag blocks from the toolboxes into the rightmost panel to control the behavior of the button.

Two scripts in the rightmost panel

Although you probably can't read the details in [Image 1](#), you can see that there are two scripts showing in the rightmost panel. (*I will show you an*

expanded screen shot of the rightmost panel later in [Image 2](#).)

For now, suffice it to say that the top script initializes the values stored in all four variables when the user clicks the green flag in the upper right of the stage in [Image 1](#). The bottom script in the rightmost panel defines the behavior of the program when the user clicks the button in the Stage area.

Variables with sliders

If you look carefully, you can tell that the top two variables that are displayed in the Stage area look different from the bottom two variables. This is because a slider has been assigned to each of the top two variables (see [Image 5](#)). This makes it possible for the user to manually set the values stored in each of these two variables by moving the thumb or pointer on the slider.

How to create a slider

To create a slider for a variable, right click on the display of the variable in the Stage area and select **slider** in the popup menu that appears.

Once you have caused a slider to appear with the variable display, you can right-click on the variable display again and select **set slider min and max** in the popup menu to specify the range of the slider. This will cause a simple dialog box to appear into which you can enter the minimum value and the maximum value and then click an OK button. In this program, I have both sliders set to a minimum value of -10 and a maximum value of +10.

Expanded view of the rightmost panel

[Image 2](#) shows an expanded view of the rightmost panel after having selected the button icon in the area immediately below the Stage in [Image 1](#)

Image 2. Expanded view of the rightmost panel.

Figure



Image 2. Expanded view of the rightmost panel.

Initialize the variable values to zero

As I mentioned earlier, the top script in [Image 2](#) sets the value of each of the four variables to zero when the user clicks the green flag. The code in that script should be completely familiar to you by now and no explanation should be necessary.

Define the behavior of the button

The bottom script in [Image 2](#) defines the behavior of the program when the button is clicked. Basically, that code says to set the value of the variable named **Sum** to the sum of the contents of the two variables named **LeftOperand** and **RightOperand** and to set the value of the variable named **Diff** to the value of the **LeftOperand** minus the value of the **RightOperand** .

Constructing the script

That seems straightforward enough. However, the process of constructing that script contains some new material, so I will walk you through the process.

[Image 3](#) shows the state of the development process immediately after

- the **set** blocks for the variables named **Sum** and **Diff** have been dragged from the **Data** toolbox to the rightmost panel and connected to the block that reads **when this sprite clicked** , and
- the **Operators** toolbox has been selected.

Image 3. Preparing to use the arithmetic operators.

Figure

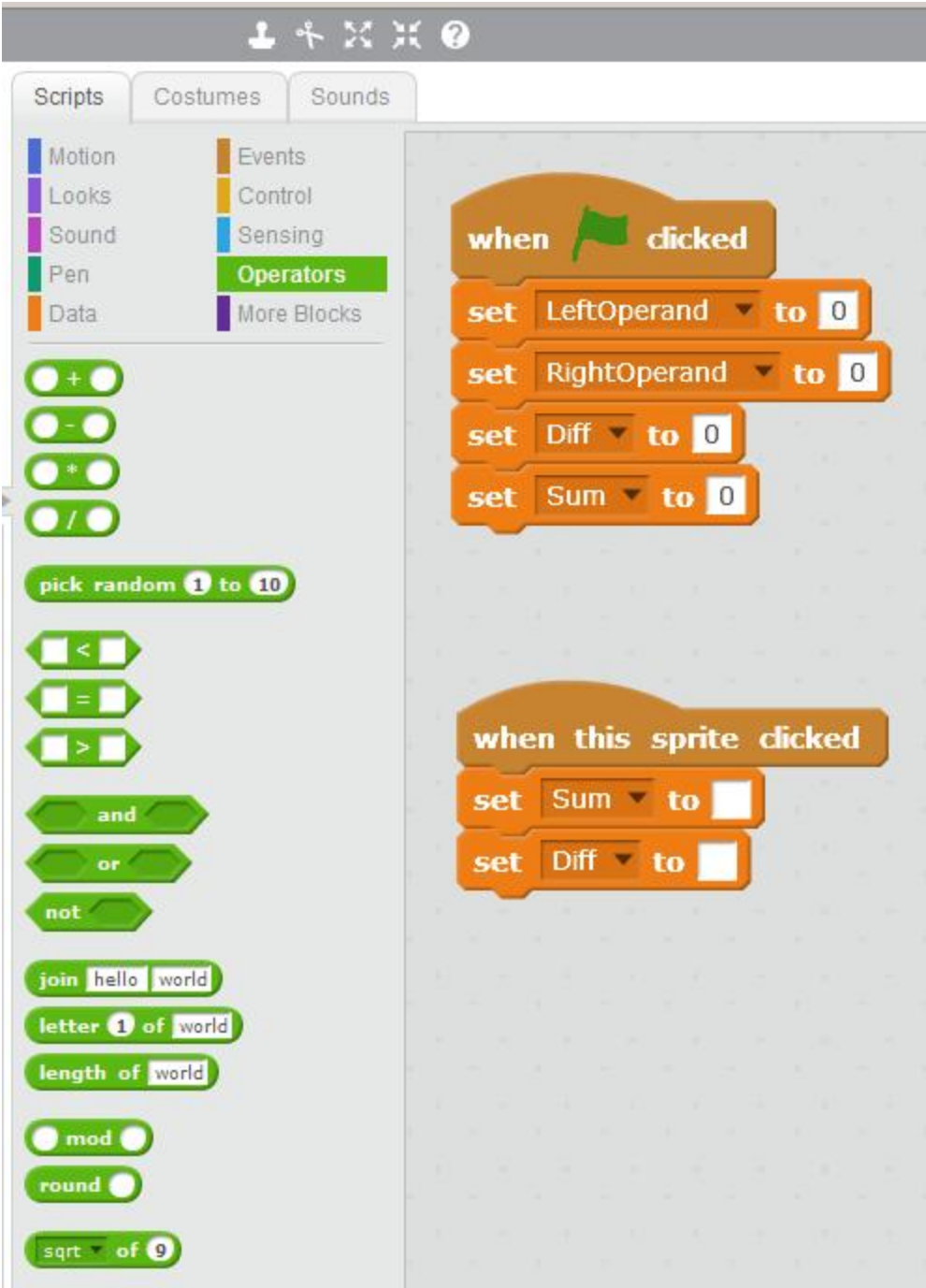


Image 3. Preparing to use the arithmetic operators.

Not exactly what we want to happen

In [Image 3](#), as it stands now, the variables named **Sum** and **Diff** will be set to a value of zero when the user clicks the button. That isn't what we want, so we have some more programming to do.

The desired behavior is for the value of the variable named **Sum** to be set to the sum of the variables named **LeftOperand** and **RightOperand** when the user clicks the button. Similarly, we want the value of the variable named **Diff** to be set to the difference between those two variables.

The Operators toolbox

The left panel in [Image 3](#) shows the blocks that are exposed when the green button labeled **Operators** is clicked.

Some of those blocks are of the correct shape to fit into the white boxes in the two bottom orange blocks in the rightmost panel. This means that we could drag any one of those blocks and drop it into one of the white boxes.

Four arithmetic operators

The top four green blocks in the left panel of [Image 3](#) are particularly interesting. They contain the following arithmetic symbols as part of their labels:

- +
- -
- *
- /

These are the symbols that are commonly used for addition, subtraction, multiplication, and division in computer programming. The top two match what you are accustomed to seeing on your hand calculator, but the bottom two probably don't match.

Use an asterisk for multiplication

Most programming languages use the `*` character instead of the **X** character for multiplication in order to keep the multiplication operator from being confused with the letter that uses the **X** symbol.

Use a forward slash for division

Insofar as division is concerned, most computer keyboards don't have a symbol that matches the symbol commonly used for division on hand calculators, so the `/` character is used instead.

More blocks of the correct shape

Before leaving the discussion of the left panel in [Image 3](#), I will explain the behavior of four additional blocks that are of the correct shape for being dropped into the white boxes in the blocks in the rightmost panel. *(The remaining blocks beyond those four deal with strings, which I will ignore for now.)*

Random numbers

The block labeled **pick random 1 to 10** lets you enter other values in place of 1 and 10. Then this block will deliver a random number within that range when it is called upon to do so. Random numbers are often used in game programs to simulate the throwing of dice or the spinning of a wheel of fortune.

The modulus

The block labeled **mod** will produce the modulus of two values when called upon to do so. The modulus of two numbers is the remainder that results from dividing one number by another number. You may remember the

remainder from when you learned to do long division in elementary school but before you learned about decimals.

Various numeric representations

The block labeled **sqrt** will deliver about a dozen different representations of numeric values, depending on what is selected from the pull-down list. This includes the square root, trigonometric functions, absolute value, etc.

Rounding a number

Finally, the block labeled **round** will round a decimal number to the nearest whole number.

Drag and drop the plus and minus operators

[Image 4](#) shows the result of dragging the top two green blocks from the **Operators** toolbox and dropping them into the white boxes in the bottom two orange variable blocks in the rightmost panel of [Image 3](#).

Image 4. Result of dropping addition and subtraction operators into variable blocks.

Figure

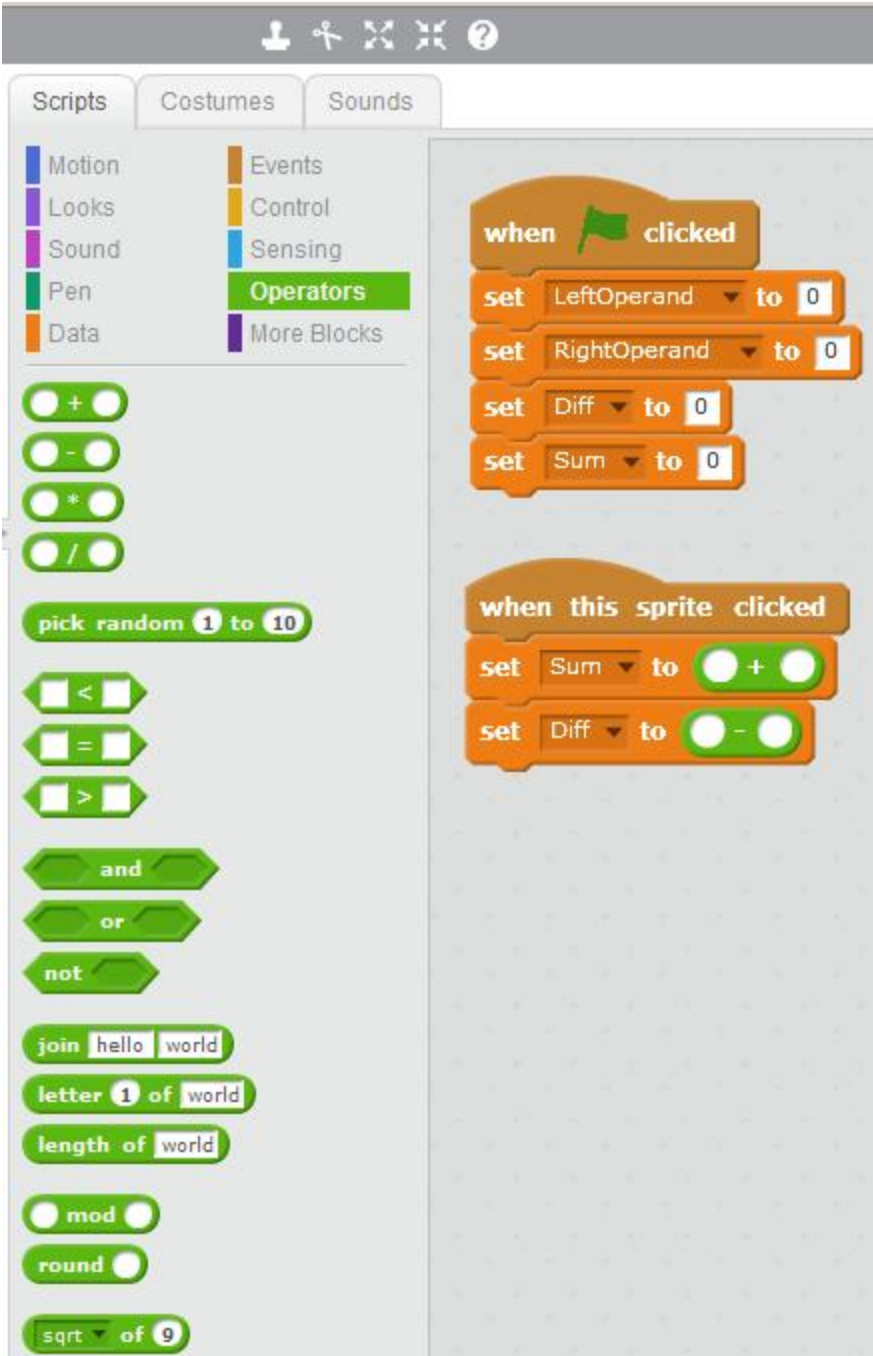


Image 4. Result of dropping addition and subtraction operators into variable blocks.

A few more steps are required

If you compare [Image 4](#) with [Image 2](#), you will see that [Image 4](#) is close to what we need but we aren't quite there yet. In order to cause the bottom two blocks in the rightmost panel in [Image 4](#) to match the bottom two blocks in [Image 2](#), we need to do the following:

- Click the orange **Data** button in the upper left of [Image 4](#) to expose the variables as shown in [Image 1](#).
- Drag the blocks for the variables named **LeftOperand** and **RightOperand** from the **Data** toolbox and drop them into the white boxes in the bottom two blocks in [Image 4](#) to make them match the bottom two blocks in [Image 2](#).

That's the solution. The upper left portion of your Stage area should now look similar to [Image 5](#). Note however that you may need to use the mouse to arrange the four variables and the button to get your Stage arranged like [Image 5](#).

Image 5. Stage area of the finished program.

Figure



Image 5. Stage area of the finished program.

Operation of the program

Once you reach this point, you can click the green flag to initialize all four variables to zero.

Then you can move the sliders back and forth to manually set the values for the variables named **LeftOperand** and **RightOperand** .

If you click immediately to the right or left of the thumb on a slider, that will cause the thumb to move a small distance in that direction.

Then when you click the button at the bottom of [Image 5](#), the variable named **Sum** will display the sum of the values of the top two variables, and the variable named **Diff** will display the value of **LeftOperand** minus the value of **RightOperand** .

An online version of this program is available

A copy of this program has been posted online for your review (see [Resources](#) for the URL) . If you don't find the program using that URL, search the Scratch site for the user named dbal.

Run the program

I encourage you to use the information provided above to write this program. Experiment with the code, making changes, and observing the results of your changes. Make certain that you can explain why your changes behave as they do.

Just for fun, move the button a little to the right and cause it to say **Ouch** for about five seconds each time you click it as shown in [Image 6](#). *Hint: See the purple button labeled **Looks** in [Image 3](#).*

Image 6. Making the button say ouch.

Figure

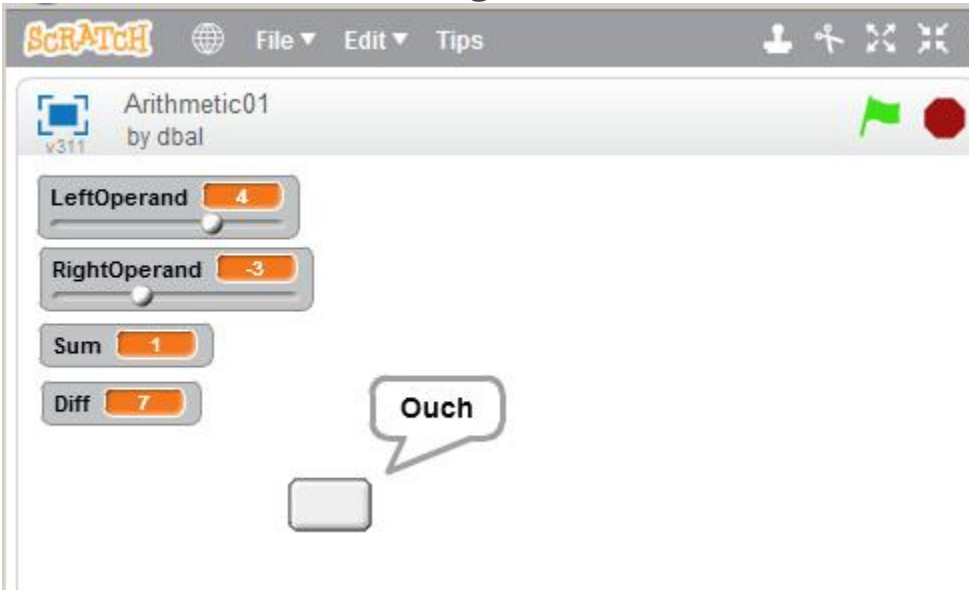


Image 6. Making the button say ouch.

I also encourage you to write the program described below.

Student programming project

Write a Scratch program named **Arithmetic02** that produces the output shown in [Image 7](#) when the user adjusts the sliders to the values shown and clicks the button. Make the word **Ouch!** appear and then go away after about five seconds.

Image 7. Output from student project program named Arithmetic02.

Figure



Image 7. Output from student project program named Arithmetic02.

A copy of this program has been posted online for your review ([see Resources for the URL](#)). If you don't find the program using that URL, search the Scratch site for the user named **dbal**.

Summary

I began by teaching you about *operators* and *operands* in general. I also taught you about *expressions* and *statements*. I gave a very brief introduction of the concept of *type*.

I presented and explained a sample Scratch program that illustrates how to use arithmetic operators in Scratch.

Finally, I provided the specifications for a student-programming project for you to write in order to demonstrate your understanding of what you learned from the first program.

Copies of both programs have been posted online for your review (see [Resources](#) for the URL) . If you don't find the program using that URL, search the Scratch site for the user named dbal.

What's next?

This module concentrated on arithmetic operators. The next few modules will deal with relational and logical operators as well as selection and loops.

Resources

- [Scratch home](#)
- [Scratch tutorials](#)
- [Scratch explanatory video](#)
- [Scratch tutorials from MIT](#)
- [Scratch tutorial - Digital Art for All](#)
- [Scratch for Budding Computer Scientists](#)
- [Learn Scratch at learnscratch.org](#)
- [Scratch Tutorial - Space Shuttle Mission STS-2020](#)
- [SCRATCH TUTORIALS ICT In Primary Education 2012-2013](#)
- [Scratch Wiki](#)
- [Scratch WikiTable of ContentsWebsite - Scratch Wiki](#)
- [Scratch WikiTable of ContentsTutorials - Scratch Wiki](#)
- [Scratch WikiTable of ContentsProgram - Scratch Wiki](#)
- [Tutorial about Variables - Scratch Wiki](#)
- [Scratch support MIT web site](#)
- [Scratch resources at Scratch.ie](#)
- [ScratchEd Resources](#)
- [Scratch project Day Dream](#)
- [Scratch project Son of String Art](#)
- [Scratch Project Scratch Tutorial](#)
- [Variable01](#) - Online version of program
- [Variable02](#) - Online version of student programming project

- [Variable03](#) - Online version of student programming project
- [IfSimple01](#) - Online version of program
- [IfWithVar01](#) - Online version of student programming project
- [Arithmetic01](#) - Online version of program
- [Arithmetic02](#) - Online version of student programming project

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name:Scr0340: Arithmetic Operators in Scratch 2.0
- File: Scr0340.htm
- Published: 05/13/13

Note: Disclaimers:

Financial : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.

I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module. In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.

Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Scr0350: Relational Operators in Scratch 2.0

The purpose of this module is to teach you how to write a Scratch program that uses the following relational operators: less than, equal to, and greater than.

Table of Contents

- [Preface](#)
 - [Viewing tip](#)
 - [Images](#)
- [General background information](#)
 - [Operator categories](#)
- [Preview](#)
 - [Program operation](#)
- [Discussion and sample code](#)
 - [A button and five variables](#)
 - [The program code](#)
 - [Initialize the variables](#)
 - [When the button is clicked...](#)
 - [An if-else block](#)
 - [Need to put a conditional clause in the pocket](#)
 - [Where are the blocks with the correct shape?](#)
 - [Will use blocks from the Operators group](#)
 - [Six blocks have the correct shape](#)
 - [Relational operators in other languages](#)
 - [Three steps at once](#)
 - [Still need to complete the conditional clause](#)
 - [Complete the relational expression](#)
 - [Completing the program](#)

- [A screen shot of the program output](#)
- [Interpretation of the results](#)
- [An online version of this program is available](#)
- [Run the program](#)
- [Student programming project](#)
 - [Operation of the program](#)
- [Summary](#)
- [What's next?](#)
- [Resources](#)
 - [General resources](#)
 - [Programs used in this collection](#)
- [Miscellaneous](#)

Preface

[Scratch 2.0](#) (*released May 9, 2013*) is the second major version of Scratch to be released during the life of the product. Among other things, it features a redesigned editor and website, and allows you to edit projects directly from your web browser.

This module (*tutorial*) is one in a collection of modules designed to help beginners of all ages (*8 and up*) learn how to create the code for computer programs using [Scratch 2.0](#). Information is provided not only for the beginners themselves but also for their parents and teachers where appropriate.

The purpose of this module is to teach you how to write a Scratch program that uses the relational operators shown below:

Note:

```
< (less than)
= (equal to)
> (greater than)
```

Viewing tip

I recommend that you open another copy of this module in a separate browser window and use the following links to easily find and view the images while you are reading about them.

Images

- [Image 1](#). Project Relational01 at startup.
- [Image 2](#). Project Relational01 output.
- [Image 3](#). Reduced screen shot of program Relational01 programming interface.
- [Image 4](#). Full size view of the programming panel for Relational01.
- [Image 5](#). Starting to use an if-else block.
- [Image 6](#). Green blocks exposed by clicking the Operators button.
- [Image 7](#). Intermediate stage in construction of if-else script.
- [Image 8](#). Screen shot of the output from the program named Relational01.
- [Image 9](#). Screen shot of the output from the program named Relational02.

General background information

First, a quick review of material from earlier modules:

- **Operators** are the action elements of a computer program. They perform actions such as adding two variables.
- **Operands** are the things that are operated on by operators. For example, variables are often the operands that are operated on by

operators.

- An **expression** is a specific combination of operators and operands that evaluates to a particular result.
- A **statement** is a specific combination of expressions.
- The equal character (=) would commonly be called the **assignment** operator in programming languages such as Java but we will see later that it is used as a *relational* operator in Scratch.
- Scratch has two types of data (*numeric and string*).
- An operator that operates on one operand is called a **unary** operator.
- An operator that operates on two operands is called a **binary** operator.
- An operator that operates on three operands is called a **ternary** operator. Scratch doesn't have any ternary operators.
- **Binary** operators in Scratch use *infix* notation. This means that the operator appears between its operands.

Operator categories

There are several different kinds of operators. The easiest way to study them is to divide them into categories such as the following

- arithmetic
- relational
- logical
- bitwise
- assignment

An earlier module explained *arithmetic* operators. This module will explain *relational* operators. Future modules will explain the other kinds of operators.

A previous module introduced you to the *selection* structure. This module will expand on that concept.

Preview

In this module, I will present and explain a Scratch program named **Relational01** . This program illustrates the use of the following **relational operators** :

Note:

```
< (less than)
= (equal to)
> (greater than)
```

The program creates five variables with the names listed below along with a button and displays them on the Stage **as shown** in [Image 1](#).

- LeftOperand - a slider
- RightOperand - a slider
- LessThan
- Equals
- GreaterThan

Image 1. Project Relational01 at startup.

Figure

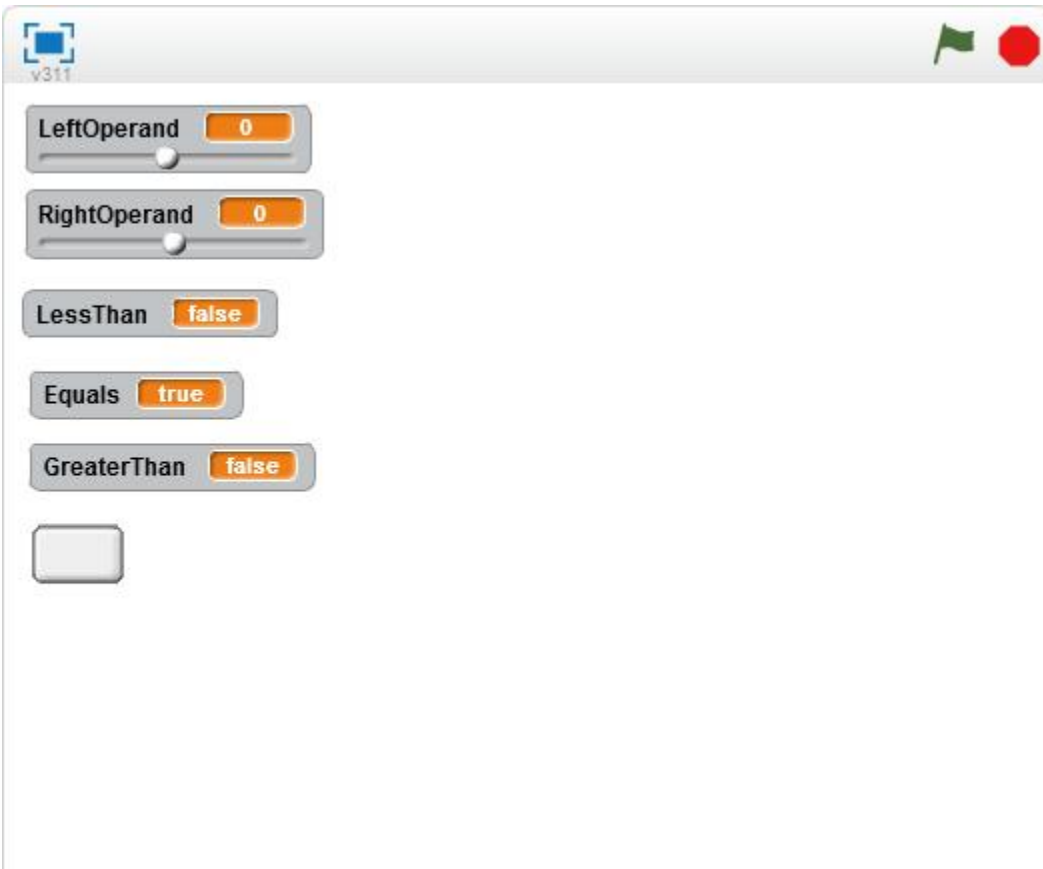


Image 1. Project Relational01 at startup.

[Image 1](#) is what you should see on the left side of your screen when you navigate to the project (see [Resources](#)) and click the large green flag. Note the values showing in each of the orange boxes. I will have more to say about them later.

[Image 2](#) shows the result of moving the thumb on the top slider one click to the left (*move to -1*) and then clicking the button at the bottom. Once again, note the values showing in the orange boxes, which I will have more to say about later.

Image 2. Project Relational01 output.

Figure

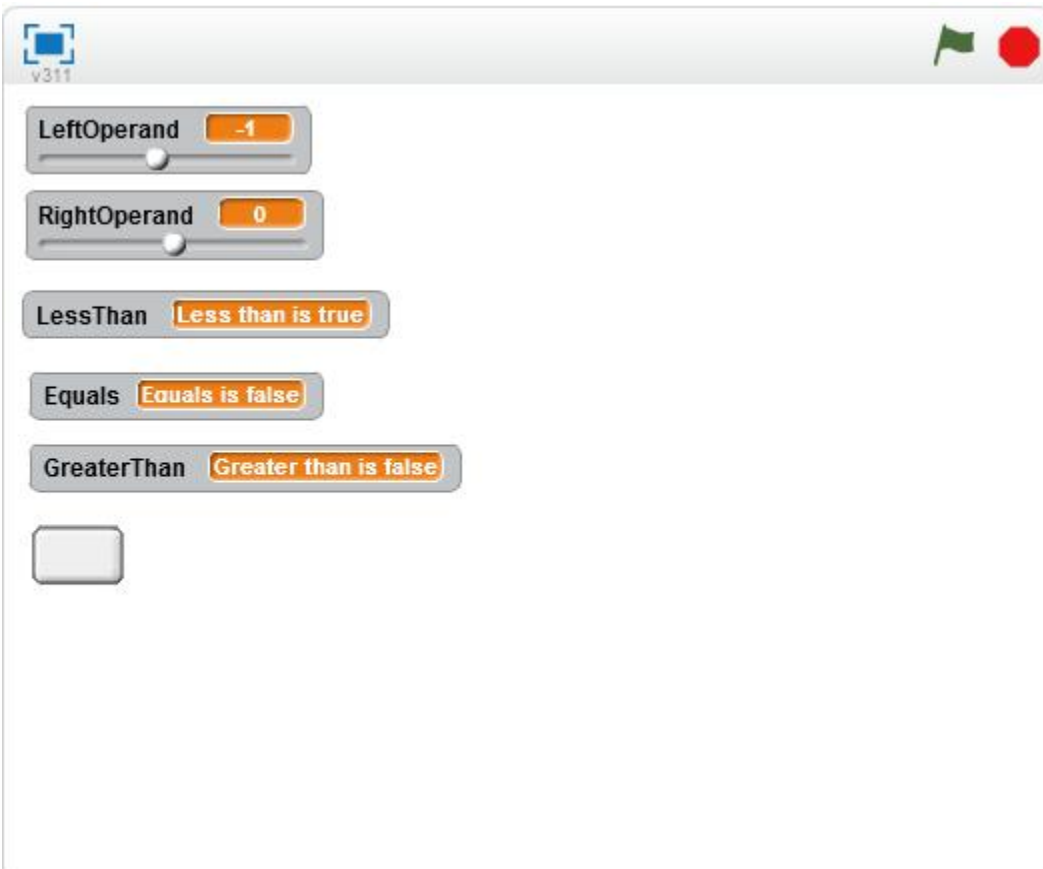


Image 2. Project Relational01 output.

[Image 3](#) shows a reduced view of the programming interface immediately after having clicked the green flag. Although it is not easy to see, the variables on the stage match [Image 1](#).

Image 3. Reduced screen shot of program Relational01 programming interface.

Figure

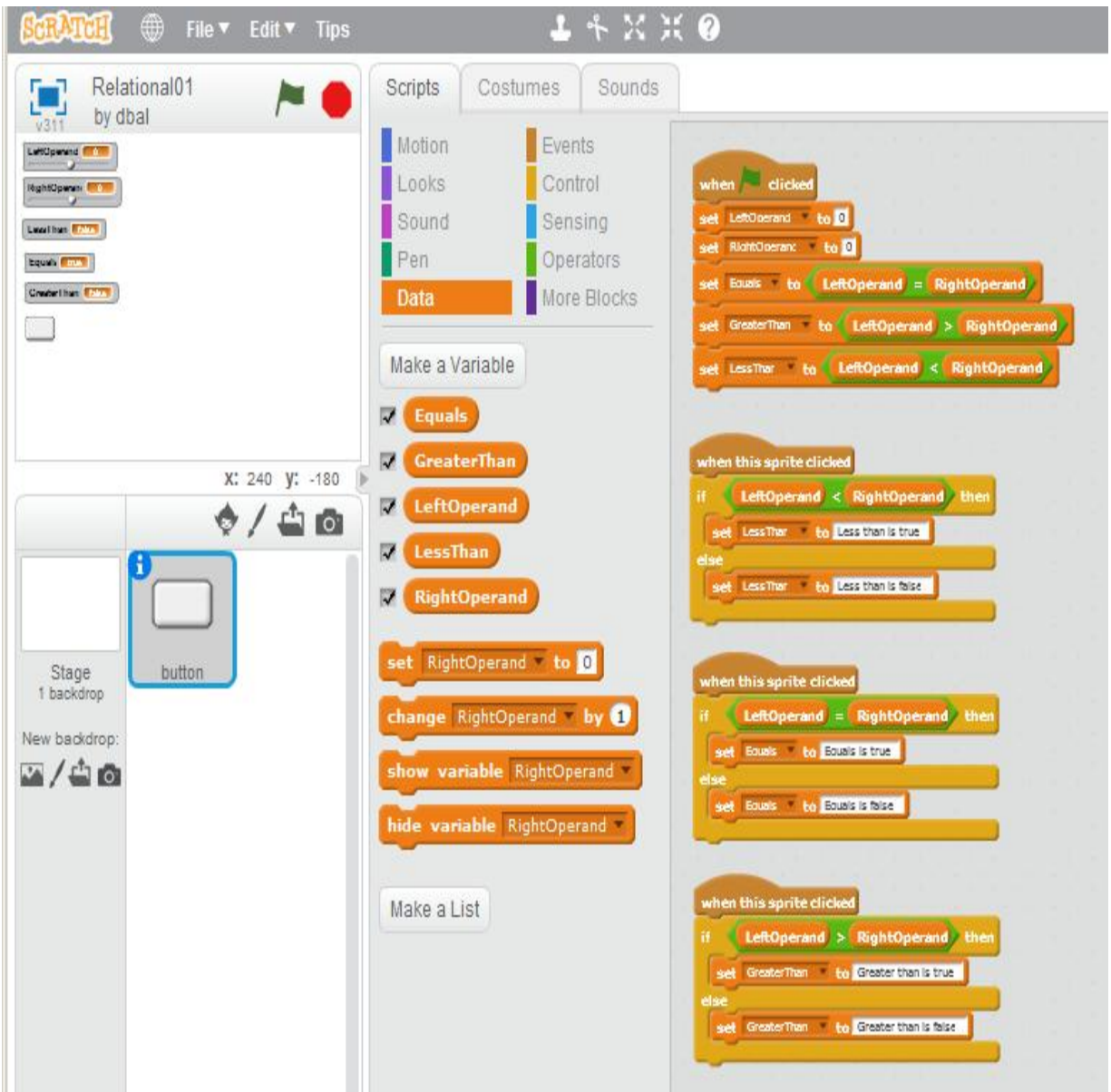


Image 3. Reduced screen shot of program Relational01 programming interface.

Program operation

When the user clicks the green flag, the values of the slider variables shown in [Image 1](#) are set to 0. Expressions containing relational operators are evaluated to set the values of the bottom three variables to the boolean values of true or false.

The user slides the two sliders to change the values of **LeftOperand** and **RightOperand**.

When the user clicks the button at the bottom of [Image 1](#), three separate event handlers on the button test the left operand against the right operand for *less than*, *equal to*, and *greater than* and display string values describing the results in the three variables having the corresponding names in [Image 1](#) and [Image 2](#). Those three event handlers are shown as the bottom three scripts in the programming panel of [Image 3](#).

Discussion and sample code

A button and five variables

As you can immediately below the stage in [Image 3](#), a button was added to this program. If you examine the toolbox and the stage in [Image 3](#), you will see that five variables were created for the program and that all five of the variables were displayed in the Stage. (*The checkboxes for all five variables were checked causing them to appear in the Stage.*) A better view of the Stage is provided in [Image 1](#).

The names of the five variables were listed [above](#). By this point, you should have no difficulty creating variables and causing them to be displayed in the Stage, so no further explanation of this aspect of the project should be necessary.

The program code

[Image 4](#) shows a full-size view of the programming panel in [Image 3](#). This panel contains the program code associated with the button.

Image 4. Full size view of the programming panel for Relational01.

Figure



Image 4. Full size view of the programming

panel for Relational01.

Initialize the variables

The top script in [Image 4](#) initializes the values of the variables named **LeftOperand** and **RightOperand** to zero.

Then it uses relational operators to evaluate three relational expressions in sequence. The results of evaluating those expressions are **boolean** values (*true or false*). Those values are used to initialize the values of the variables named **Equals** , **GreaterThan** , and **LessThan** .

The relational expressions are constructed by dropping the orange variable names on either side of the [relational operators](#) in the green blocks with the pointed ends.

[Image 1](#) shows the results of evaluating those expressions. When both operands have a value of zero, the results are:

- LessThan = false
- Equals = true
- GreaterThan = false

When the button is clicked ...

Each of the bottom three scripts in [Image 4](#) are executed when the user clicks the button in the Stage area in [Image 3](#) . These three scripts are very similar. However, they use three different *relational operators* to compare the values of the left and right operands and they set the values in three different output variables accordingly.

An if-else block

I will walk you through the construction of one of the bottom three scripts shown in [Image 4](#) with a detailed explanation of each step in the process. You should be able to extend that explanation to the other two scripts.

[Image 5](#) shows the result of

- Selecting the **button** sprite immediately below the stage in [Image 3](#).
- Selecting the **Events** toolbox.
- Dragging the block labeled **when this sprite clicked** into the programming panel.
- Selecting the **Control** toolbox.
- Dragging an **if-else** block from the toolbox into the programming panel.
- Connecting the **if-else** block to the block that is labeled **when this sprite clicked**.

*By the way, in case I forgot to tell you before, you can set the name of the sprite to **button** by selecting the **Costumes** tab above the toolbox selector panel and entering the name in the text field.*

Image 5. Starting to use an if-else block.

Figure

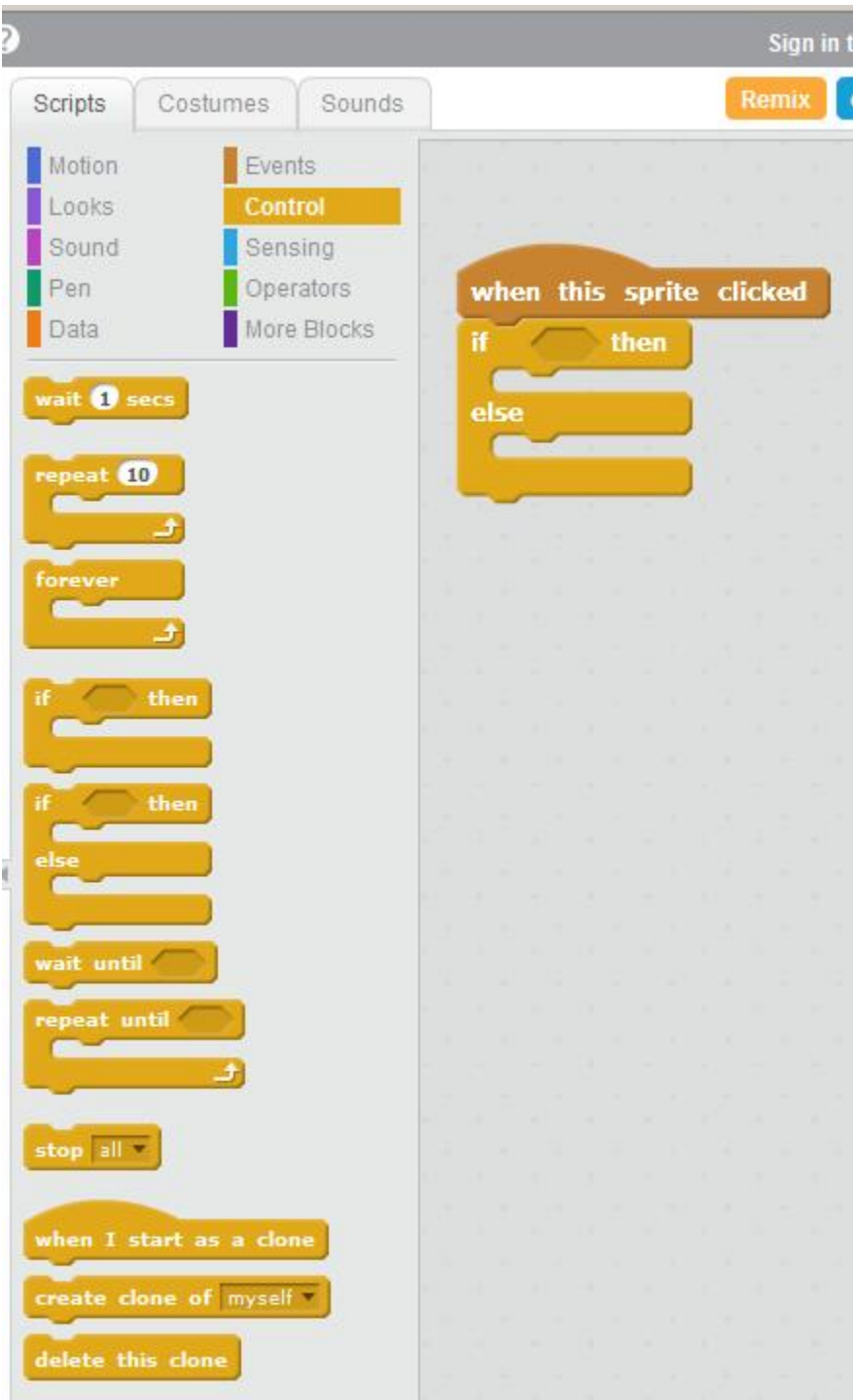


Image 5. Starting to use an if-else block.

Need to put a conditional clause in the pocket

Note the depressed pocket immediately to the right of the word **if** on the **if-else** block. We must drop another block of the correct shape into this pocket. The code in that block must evaluate to either true or false.

If that code evaluates to true, the code that we will insert into the mouth of the block immediately below the word **if** will be executed.

If the code that we drop into the pocket evaluates to false, the code that we insert into the mouth of the block immediately below the word **else** will be executed.

Where are the blocks with the correct shape?

There are two buttons (*possibly more*) that we can click in the upper left in [Image 5](#) that will expose blocks with shapes matching the pocket immediately to the right of the word **if** in [Image 5](#):

- Sensing
- Operators

Will use blocks from the Operators group

In this module, we will use three of the blocks that are exposed by clicking the green **Operators** button as shown in [Image 6](#).

Image 6. Green blocks exposed by clicking the Operators button.

Figure

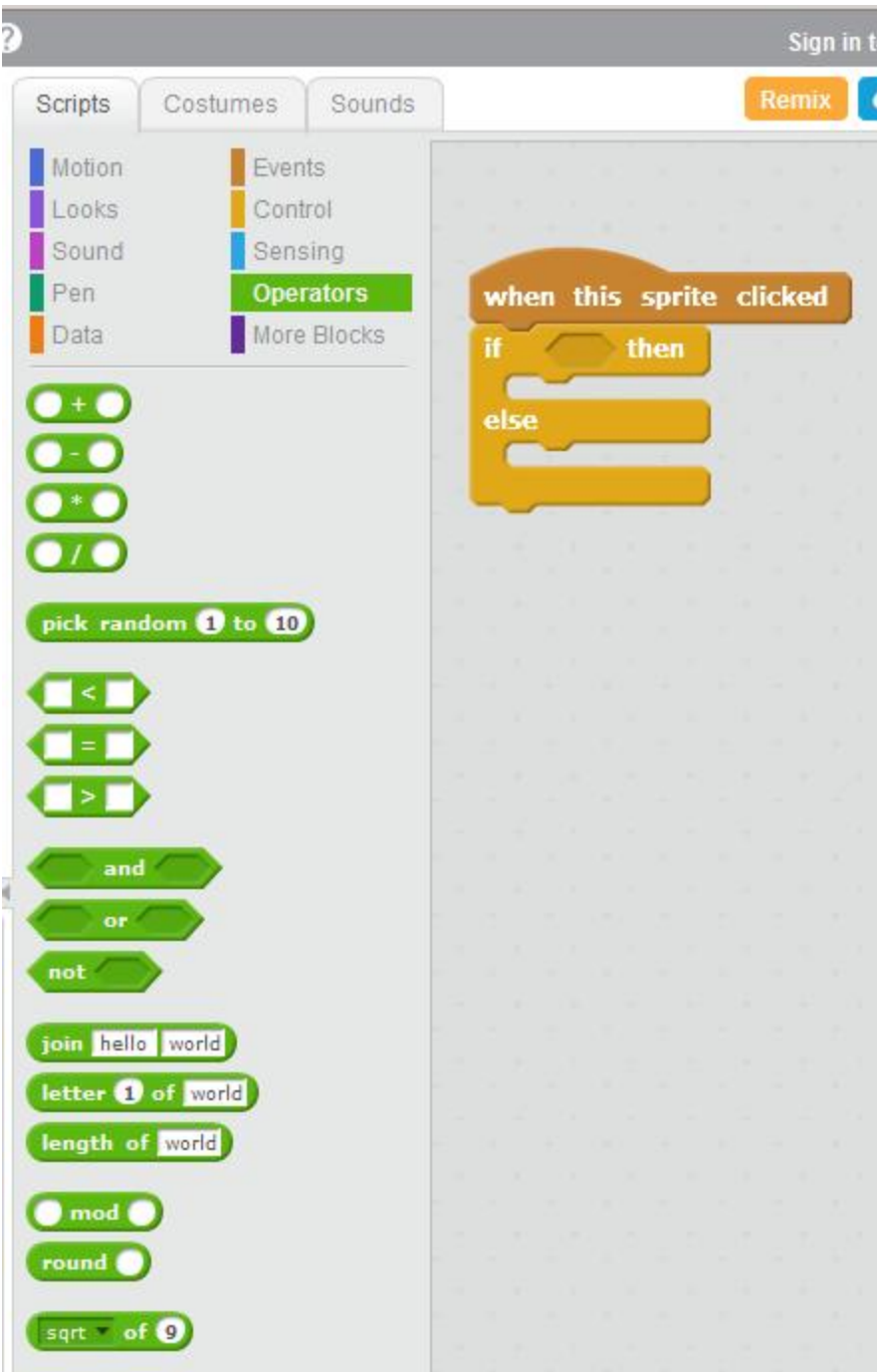


Image 6. Green blocks exposed by clicking the Operators button.

Six blocks have the correct shape

As you can see in [Image 6](#), there are six blocks of the correct shape exposed by clicking the **Operators** button. Those blocks are identified as follows:

Note:

```
< (less than)
= (equal to)
> (greater than)
and
or
not
```

We will refer to the first three blocks above as ***relational operators*** . (*They are used to evaluate the relationship between two values.*) We will refer to the other three blocks in the above list as ***logical operators*** , and will deal with them in a future module.

Relational operators in other languages

Many other modern programming languages also include the following three relational operators:

Note:

```
>= (greater than or equal)
<= (less than or equal)
```

```
!= (not equal)
```

Although these operators are a great convenience, they are not essential. We can get by with only the three that are provided by Scratch.

Also, in many other programming languages, a pair of equal characters (==) is used for the "*equal to*" operator instead of the single equal character (=) used in Scratch.

Three steps at once

[Image 7](#) shows the result of three more steps in the development of the program:

1. Dragging the green **less than** (*indicated by a left angle bracket*) block and dropping it into the pocket immediately to the right of the word **if** .
2. Dragging the orange block labeled **set...to** for the variable named **LessThan** and inserting it immediately below the word **if** . I also set its literal value to "Less than is true".
3. Dragging another copy of the orange block labeled **set...to** for the variable named **LessThan** and inserting it immediately below the word **else** . *I also set its literal value to "Less than is false".*

Image 7. Intermediate stage in construction of if-else script.

Figure

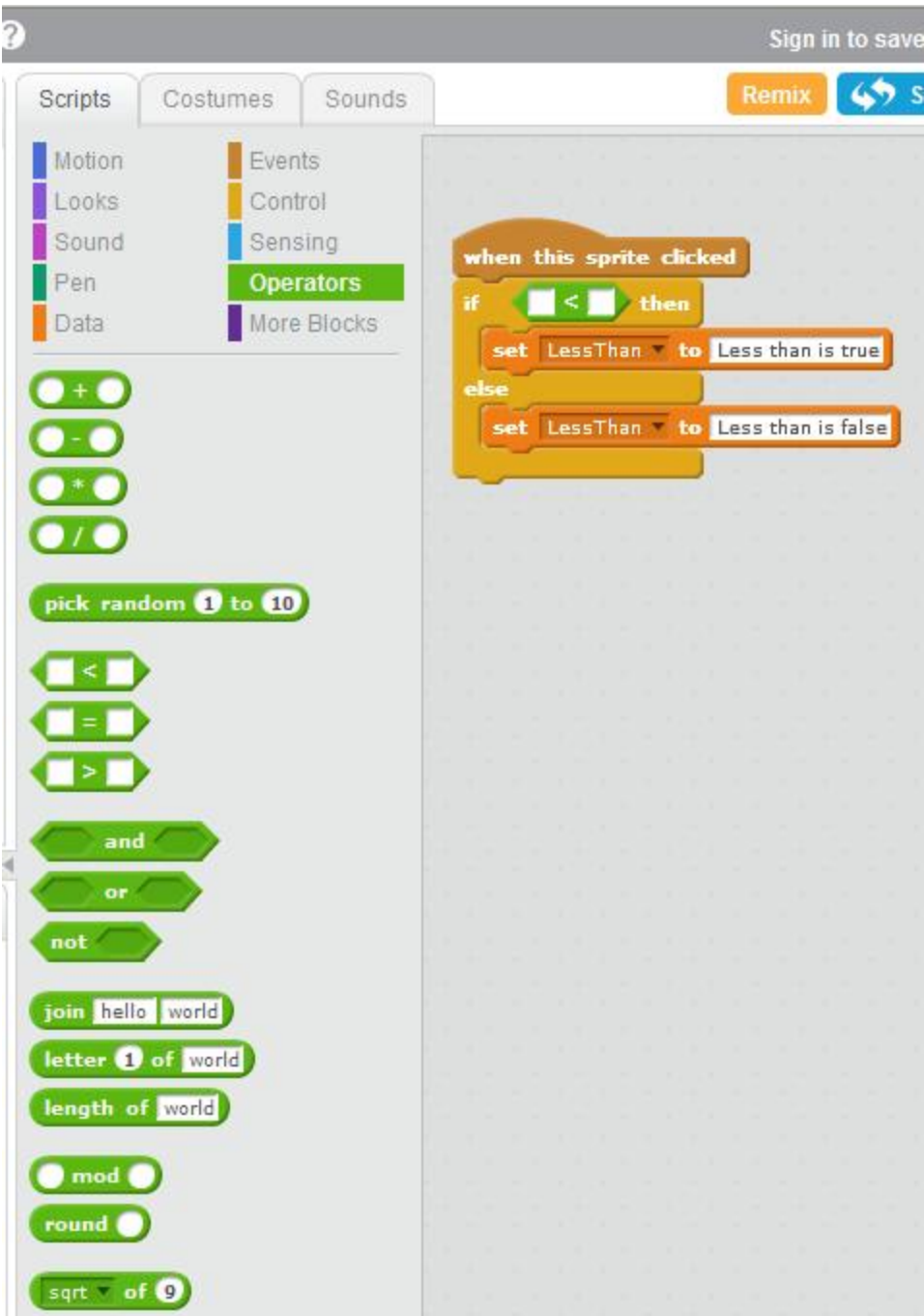


Image 7. Intermediate stage in construction of if-else script.

Still need to complete the conditional clause

The script in [Image 7](#) includes the action to be taken if the relational expression following the word **if** evaluates to true. It also includes the action to be taken if the relational expression evaluates to false. (*This relational expression is often referred to as a conditional clause.*) .

However, we still haven't completed the conditional clause in [Image 7](#). We don't want to test for the relationship between the literal values blank and blank as shown in [Image 7](#). Instead we want to test the relationship between the values of the variables named **LeftOperand** and **RightOperand** as shown in the corresponding script in [Image 4](#).

Complete the relational expression

The next and final step for constructing this script is to:

- Click the **Data** button to expose the variables as shown in [Image 3](#).
- Drag the block for the variable named **LeftOperand** and drop it in the white box immediately to the left of the left angle bracket operator in [Image 7](#).
- Drag the block for the variable named **RightOperand** and drop it in the white box immediately to the right of the left angle bracket operator in [Image 7](#).

Once we do that, we will have finished the construction of the second script from the top in [Image 4](#).

Completing the program

Following that, we need to go through essentially the same process to construct the bottom two scripts in [Image 4](#), using the other two relational operators in [Image 6](#) along with the variables named **LeftOperand** and **RightOperand** .

A screen shot of the program output

[Image 8](#) shows a screen shot of the Stage for a given set of values for the variables named **LeftOperand** and **RightOperand**.

Image 8. Screen shot of the output from the program named Relational01.

Figure

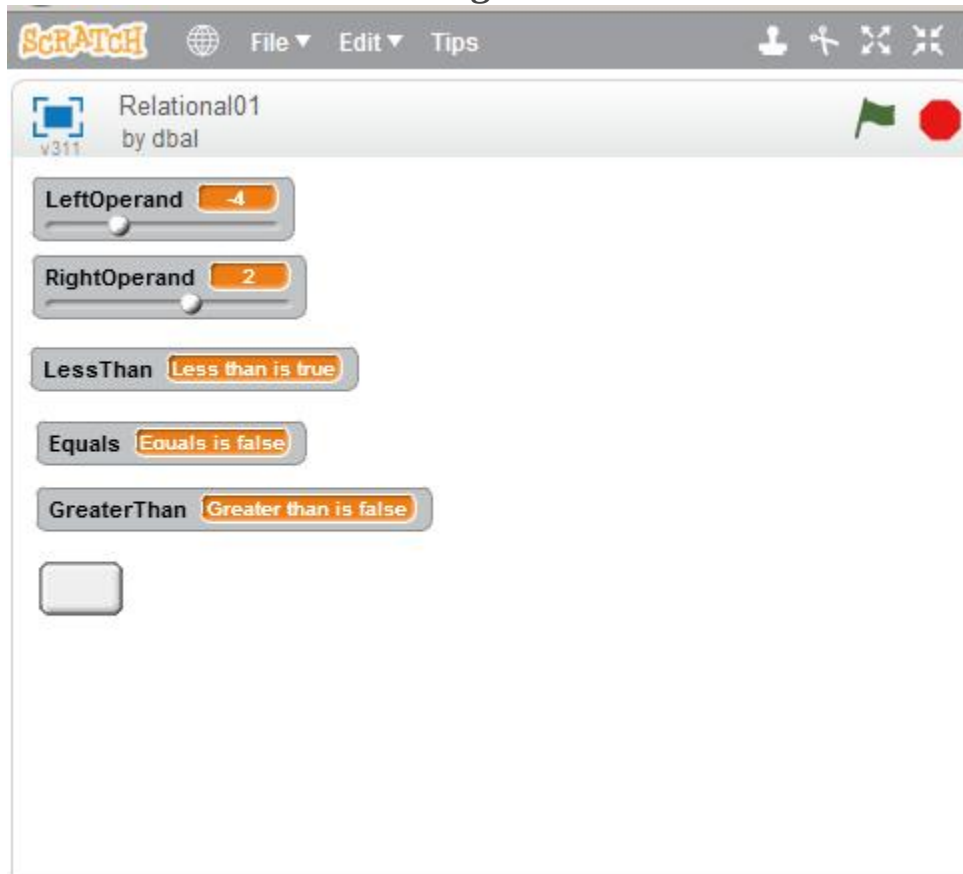


Image 8. Screen shot of the output from the program named Relational01.

Interpretation of the results

The text showing in the bottom three variables tells us that for these values, the relationships among the values are as follows:

The value of the left operand (-4) is less than the value of the right operand (2).

The value of the left operand (-4) is not equal to the value of the right operand (2).

The value of the left operand (-4) is not greater than the value of the right operand (2).

If you move the sliders to change the values in the left and right operand variables and then click the button, you are likely to get different results.

An online version of this program is available

A copy of this program has been posted online for your review (see [Resources](#) for the URL) . If you don't find the program using that URL, search the Scratch site for the user named **dbal** .

Run the program

I encourage you to use the information provided above to write this program. Experiment with the code, making changes, and observing the results of your changes. Make certain that you can explain why your changes behave as they do.

Just for fun, see if you cause a short drum roll to be produced by your computer speakers each time the button is clicked.

I also encourage you to write the program described below.

Student programming project

Write a project named ***Relational02*** that meets the following specifications:

This program illustrates the use of the following *relational operators* in addition to arithmetic operators:

Note:

< (less than)
= (equal to)
> (greater than)

This is an upgrade to the program named **Relational01** .

The program creates the following six variables and a button and displays them on the stage as shown in [Image 9](#):

- LeftOperand - a slider
- RightOperand - a slider
- Offset - a slider
- LessThan
- Equals
- GreaterThan

Image 9. Screen shot of the output from the program named Relational02.

Figure

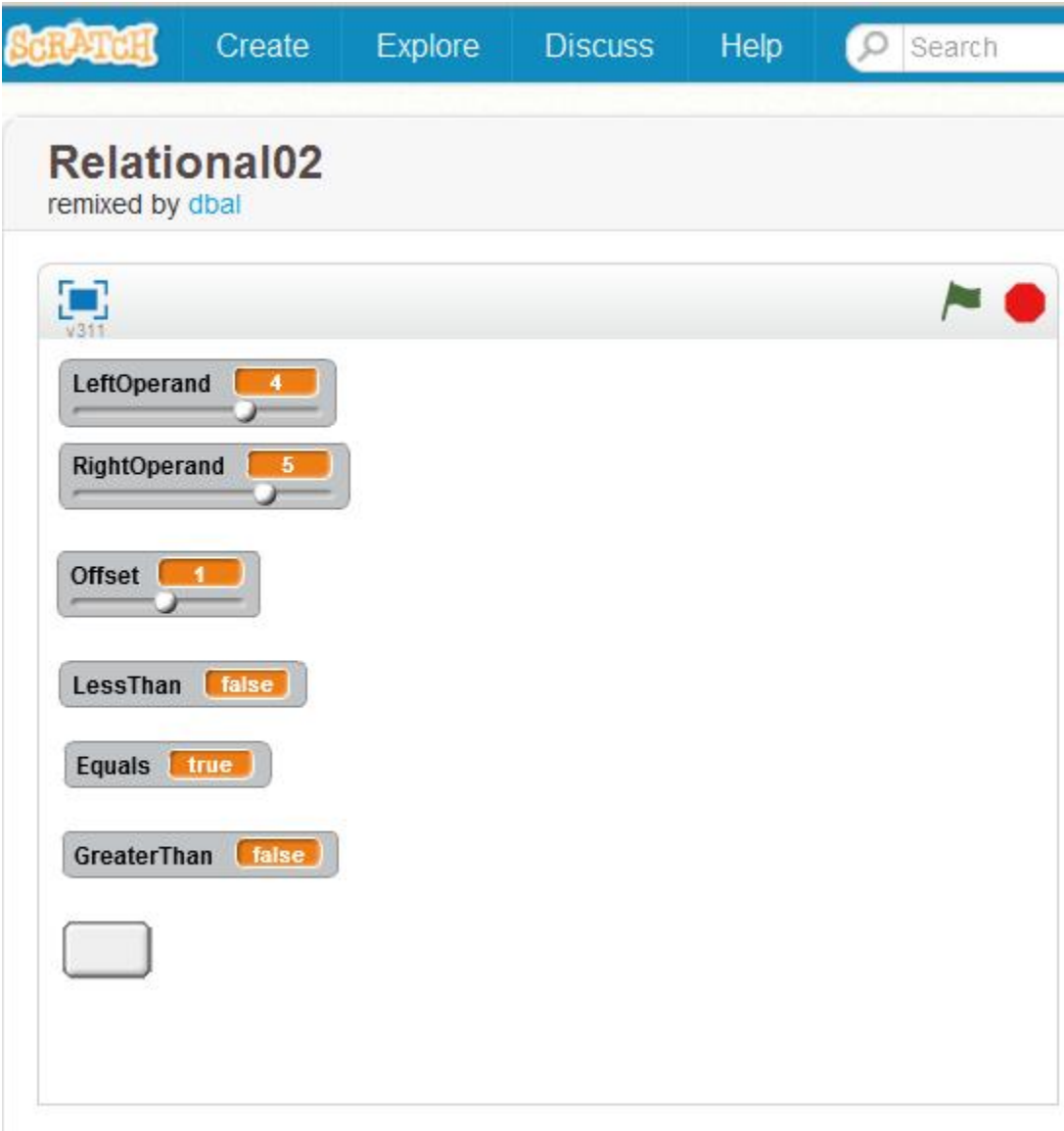


Image 9. Screen shot of the output from the program named Relational02.

Operation of the program

When the user clicks the green flag, the values of all six variables shown in [Image 9](#) are set to 0.

The user slides the three sliders to set the values of **LeftOperand** , **RightOperand** , and **Offset** .

When the user clicks the button, three separate event handlers on the button test the *sum of the left operand and the offset* against the right operand for *less than* , *equal to* , and *greater than* and display the results in the three variables having the corresponding names as either true or false.

Make sure that your output matches the output shown in [Image 9](#) for the slider values shown in [Image 9](#).

A copy of this program has been posted online for your review (see [Resources](#) for the URL) . If you don't find the program using that URL, search the Scratch site for the user named dbal.

Summary

I began by providing a brief review of material from previous modules.

Then I presented and explained the detailed steps required to write a Scratch program that uses the following relational operators:

Note:

```
< (less than)
= (equal to)
> (greater than)
```

Finally, I provided the specifications for a student-programming project for you to demonstrate your understanding of what you learned from the first program and from earlier modules.

Copies of both programs have been posted online for your review (see [Resources](#) for the URL) . If you don't find the program using that URL, search the Scratch site for the user named dbal.

What's next?

The previous module concentrated on *arithmetic* operators. This module explained the use of *relational* operators. The next module will explain the use of *logical* operators. Future modules using the Scratch programming language will continue to deal with *selection* and will also deal with *loops* .

Resources

General resources

- [Scratch home](#)
- [Scratch tutorials](#)
- [Scratch explanatory video](#)
- [Scratch tutorials from MIT](#)
- [Scratch tutorial - Digital Art for All](#)
- [Scratch for Budding Computer Scientists](#)
- [Learn Scratch at learnscratch.org](#)
- [Scratch Tutorial - Space Shuttle Mission STS-2020](#)
- [SCRATCH TUTORIALS ICT In Primary Education 2012-2013](#)
- [Scratch Wiki](#)
- [Scratch WikiTable of ContentsWebsite - Scratch Wiki](#)
- [Scratch WikiTable of ContentsTutorials - Scratch Wiki](#)
- [Scratch WikiTable of ContentsProgram - Scratch Wiki](#)
- [Tutorial about Variables - Scratch Wiki](#)
- [Scratch support MIT web site](#)
- [Scratch resources at Scratch.ie](#)
- [ScratchEd Resources](#)
- [Scratch project Day Dream](#)
- [Scratch project Son of String Art](#)
- [Scratch Project Scratch Tutorial](#)

Programs used in this collection

- [Variable01](#) - Online version of program
- [Variable02](#) - Online version of student programming project
- [Variable03](#) - Online version of student programming project
- [IfSimple01](#) - Online version of program
- [IfWithVar01](#) - Online version of student programming project
- [Arithmetic01](#) - Online version of program
- [Arithmetic02](#) - Online version of student programming project
- [Relational01](#) - Online version of program
- [Relational02](#) - Online version of student programming project

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Scr0350: Relational Operators in Scratch 2.0
- File: Scr0350.htm
- Published:05/16/13

Note: Disclaimers:

Financial : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.

I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module. In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales

nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.

Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Scr0360: Logical Operators in Scratch 2.0

The purpose of this module is to teach you about logical operators and truth tables. You will also learn how to write a Scratch program that illustrates the use of the logical and, or, and not operators when used in conjunction with relational operators.

Table of Contents

- [Preface](#)
 - [Viewing tip](#)
 - [Images](#)
- [General background information](#)
- [Preview](#)
 - [A real-world example](#)
 - [Formulate this using logical operator concepts](#)
 - [The not operator](#)
 - [The reality - logic in a formal sense](#)
 - [A sample program named Logical01](#)
 - [A button and seven variables](#)
 - [Event handlers](#)
 - [Program operation](#)
 - [Meaningful variable names](#)
 - [Insufficient space](#)
 - [A truth table](#)
 - [What does this mean?](#)
 - [The and operation](#)
 - [The inclusive or operation](#)
 - [A practical example](#)
 - [Evaluation against trial values](#)
 - [Let's walk through this](#)

- [Can become very complex](#)
 - [Program output](#)
- [Discussion and sample code](#)
 - [Compare Image G with Image C](#)
 - [No detailed explanation of the construction is required](#)
 - [Program behavior at runtime](#)
 - [Order of execution of scripts](#)
 - [Not important in this program](#)
 - [Evaluate the relational-logical expressions and take appropriate action](#)
 - [An online version of this program is available](#)
- [Run the program](#)
- [Student programming projects](#)
 - [The program named Logical02](#)
 - [The program named Logical03](#)
- [Summary](#)
- [What's next?](#)
- [Resources](#)
 - [General resources](#)
 - [Programs used in this collection](#)
- [Miscellaneous](#)

Preface

[Scratch 2.0](#) (*released May 9, 2013*) is the second major version of Scratch to be released during the life of the product. Among other things, it features a redesigned editor and website, and allows you to edit projects directly from your web browser.

This module (*tutorial*) is one in a collection of modules designed to help beginners of all ages (*8 and up*) learn how to create the code for computer

programs using [Scratch 2.0](#). Information is provided not only for the beginners themselves but also for their parents and teachers where appropriate.

The purpose of this module is to teach you about *logical operators* and *truth tables*. You will also learn how to write a Scratch program that illustrates the use of the logical **and**, **or**, and **not** operators when used in conjunction with relational operators.

You learned about *relational operators* in an earlier module.

Viewing tip

I recommend that you open another copy of this document in a separate browser window and use the following links to easily find and view the images while you are reading about them.

Images

- [Image A](#). Reduced screen shot of the programmer interface for the program named Logical01.
- [Image B](#). A simple truth table.
- [Image C](#). A truth table for the logic in the program named Logical01.
- [Image D](#). Program output for row 1 in Image C.
- [Image E](#). Program output for row 2 in Image C.
- [Image F](#). Program output for row 3 in Image C.
- [Image G](#). Programming panel for the program named Logical01.
- [Image H](#). Program output from the program named Logical02.

General background information

There are many different kinds of operators. The easiest way to study them is to divide them into categories such as the following

- arithmetic
- relational
- logical
- bitwise
- assignment

Earlier modules explained arithmetic operators and relational operators. This module will explain logical operators. Future modules will deal with the remaining kinds of operators.

Preview

In this module, I will present and explain a Scratch program named **Logical01** . This program illustrates the use of the **and** , **or** , and **not** operators in Scratch. These are called *logical* operators.

In this module, I will provide an explanation of *logical* operators and *truth tables* . Then I will present and explain a Scratch program that illustrates the use of the logical **and** , **or** , and **not** operators when used in conjunction with *relational* operators. You learned about relational operators in an earlier module.

Finally, I will provide the specifications for two programming projects for you to demonstrate your understanding of what you learned from the first program and from earlier modules. One of the programming projects is relatively difficult requiring an understanding of [DeMorgan's theorem](#) .

Copies of all three programs will be posted online for your review (see [Resources](#) for the URL) . If you don't find the program using that URL, search the Scratch site for the user named **dbal** .

A real-world example

Logical operators can be confusing at first glance, but perhaps a real world example will help to eliminate some of that confusion. For many years in this country, we have had a movie rating system designed to control the

viewing of objectionable content by young people. According to [Wikipedia](#), the **R** rating means something like the following:

"Restricted - Under 17 requires accompanying parent or adult 17 or older with photo ID"

Formulate this using logical operator concepts

In most cases, this means that a person will be allowed entry into the theatre to view the movie if:

- The person has the price of admission ***and*** is older than 16, ***or***
- The person has the price of admission ***and*** is under 17 ***and*** is accompanied by an adult who also has the price of admission ***and*** the adult can show a photo ID.

Note the use of the boldface Italicized words ***or*** and ***and*** in the above example.

The ***not*** operator

The ***not*** operator is a little more difficult to explain in terms of real world examples. At this point, suffice it to say that the application of the ***not*** operator causes something that is true to become false and causes something that is false to become true. For example, if someone is older than 16, that person is ***not*** 16 or younger.

The reality - logic in a formal sense

The reality is that you have been dealing with this kind of logic most of your life. However, unless you had a specific reason to do so, such as being

a computer programmer, a computer engineer, or perhaps an attorney, you may never have thought about it in a formal sense. The time to think about it in a formal sense has arrived.

A sample program named Logical01

A button and seven variables

This program creates a button along with the following seven variables.

Note:

```
A (a slider variable)
B (a slider variable)
C (a slider variable)
D (a slider variable)
A<B and C<D
not(A<B and C<D)
A<B or C<D
```

The program displays the button and the variables on the Stage and initializes all of the variables to zero when the user clicks the green flag as shown in [Image A](#).

Image A. Reduced screen shot of the programmer interface for the program named Logical01.

Figure

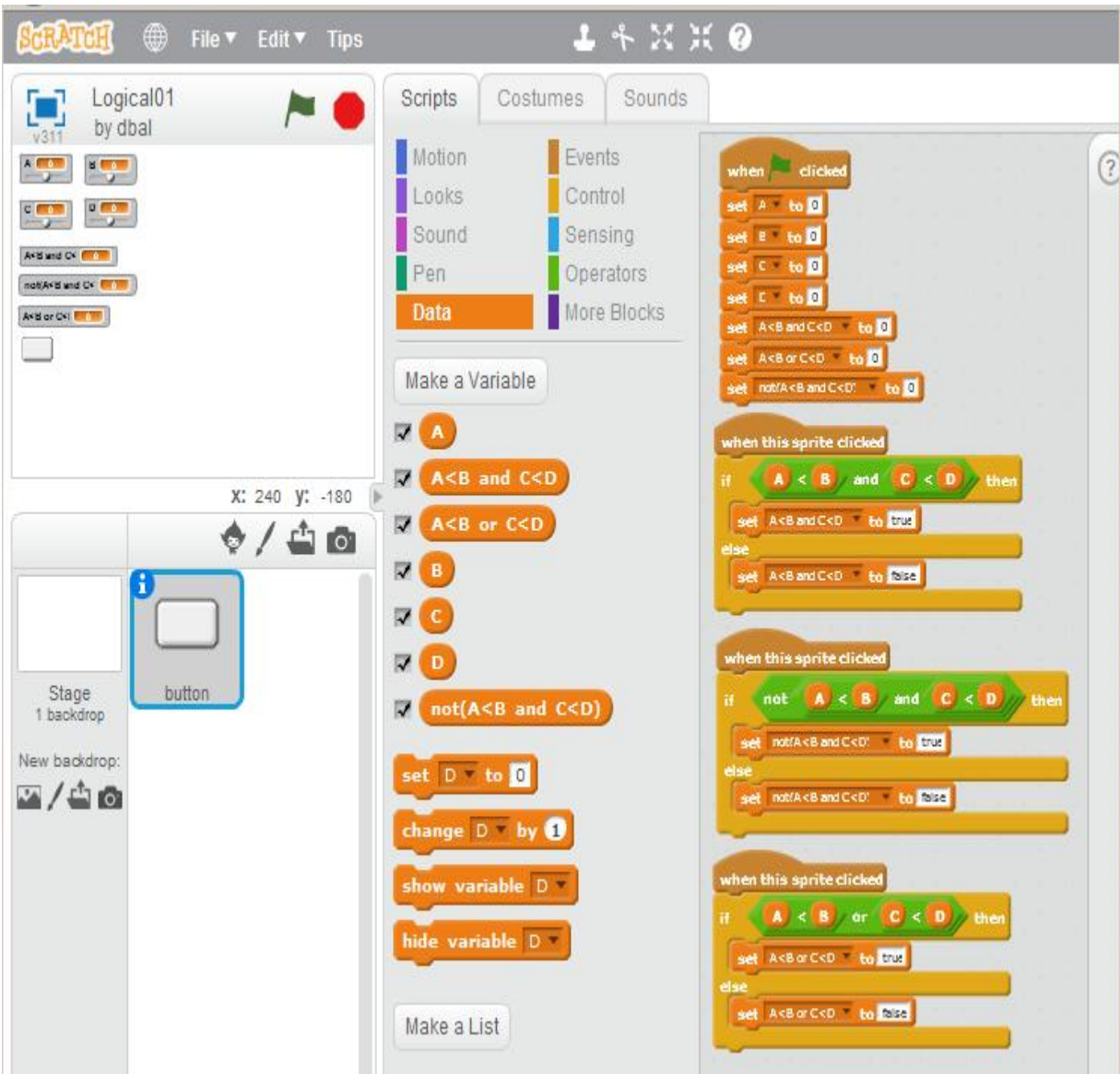


Image A. Reduced screen shot of the programmer interface for the program named Logical01.

The range of each slider variable is set to -10 to 10.

Event handlers

Three event handlers, shown as the bottom three scripts in the programming panel in [Image A](#), are written to fire each time the button is clicked.

Program operation

At runtime, the user adjusts the value of each of the slider variables.

When the user clicks the button, each event handler evaluates a *relational-logical* expression (*an expression containing both relational and logical operators*) . If the expression evaluates to true, a value of true is displayed in one of the bottom three variables on the stage in [Image A](#) . If the expression evaluates to false, a value of false is displayed in that variable.

Meaningful variable names

In an earlier module, I told you that you should always strive to use meaningful variable names. At this point, you are probably thinking that variables named **A** , **B** , **C** , and **D** aren't very meaningful, and if so, your thinking is correct. However, there is a practical reason that I didn't use meaningful variable names in this case.

Insufficient space

Really meaningful variable names usually require about five or more characters. However, I have limited display width in this presentation format. If I had used longer names for the variables in this program, the expressions containing those variable names would have been quite long, and would have made it difficult for me to fit [Image A](#) into this presentation format.

You will note, however, that the bottom three variable names on the stage in [Image A](#) (*also see the variables in the Data toolbox in [Image A](#)*) are meaningful. In fact they are probably more meaningful than would be the

case in Java because Java doesn't allow the space character, the *left angle bracket* character, or the "(" character to be included in a variable name.

A truth table

According to Wikipedia:

*"A [truth table](#) is a mathematical table used in logic -- specifically in connection with Boolean algebra, boolean functions, and propositional calculus -- to compute the functional values of logical expressions on each of their functional arguments, that is, on each combination of values taken by their logical variables. In particular, **truth tables** can be used to tell whether a propositional expression is true for all legitimate input values, that is, logically valid."*

What does this mean?

The concept of a truth table is simpler than the above quotation might suggest. [Image B](#) is a simple truth table that shows the result of evaluating four logical expressions for all combinations of the values of two variables, **A** and **B**, where each variable can only take on the values of *true* and *false*. Image B. A simple truth table.

Figure

A	B	(A and B)	(A or B)	not(A and B)	not(A or B)
True	True	True	True	False	False
True	False	False	True	True	False
False	True	False	True	True	False
False	False	False	False	True	True

Image B. A simple truth table.

The *and* operation

If both **A** *and* **B** are true, then the expression (**A and B**) evaluates to true. Otherwise, it evaluates to false. Correspondingly, if (**A and B**) is true, then **not(A and B)** is false. Otherwise, it is true. Applying the **not** operator to a boolean value is often referred to as getting the *complement* of that value.

The *inclusive or* operation

The operation that I am getting ready to discuss is known as an *inclusive or* operation. There is also an *exclusive or* operation that I won't discuss in this module. That will be a topic for a much more advanced module.

If either **A** *or* **B** are true, then the expression (**A or B**) evaluates to true. Otherwise, it evaluates to false. Correspondingly, if (**A or B**) is true, then **not(A or B)** is false. Otherwise it is true.

A practical example

[Image C](#) is a truth table that shows some of the possible results produced by the logic used in the program named **Logical01**.

Image C. A truth table for the logic in the program named Logical01.

Figure

	A	B	C	D	((A < B)and(C < D))	not((A < B)and(C < D))	((A < B)or(C < D))
1	2	3	-2	-1	True	False	True
2	2	2	-2	-1	False	True	True
3	2	2	-2	-2	False	True	False

Image C. A truth table for the logic in the program named Logical01.

Evaluation against trial values

On any given row in [Image C](#), the four columns labeled **A** , **B** , **C** , and **D** show trial values that can be assigned to the corresponding variables having the same names in the program. Each of the rightmost three columns on that row shows the result of evaluating the *relational-logical* expression at the top of the column.

Let's walk through this

Let's walk across the row labeled **2** . Although the value of **C** is algebraically less than the value of **D** , the value of **A** is not less than the value of **B** because the two have equal values. As a result, the expression **((A Lt B)and(C Lt D))** evaluates to false.

Note that for publishing reasons that are too complicated to explain here, it is sometimes necessary for me to substitute the letters "Lt" in the text for the left angle bracket.

Both operands of an **and** operator must be true for the entire expression to be true. Correspondingly, the complement of that value shown in the next column is true.

Even though the value of **A** is not less than the value of **B** for row 2, the value of **C** is less than the value of **D**. Therefore, the value of **((A Lt B)or(C Lt D))** is true. Only one operand of an inclusive **or** operator must be true for the entire expression to be true.

Can become very complex

Although not illustrated here, extremely complex logical expressions can be constructed by using parentheses for grouping terms and applying a combination of **and** operators, **or** operators, and **not** operators.

Program output

[Image D](#), [Image E](#), and [Image F](#) show screen shots of the Stage in [Image A](#) for different combinations of values for the slider variables **A**, **B**, **C**, and **D**. The values for the four variables in each of the three images correspond to the values assigned to the corresponding columns in [Image C](#).

Compare the output values in each image with the values shown in the corresponding columns in [Image C](#).

Image D. Program output for row 1 in Image C.

Figure

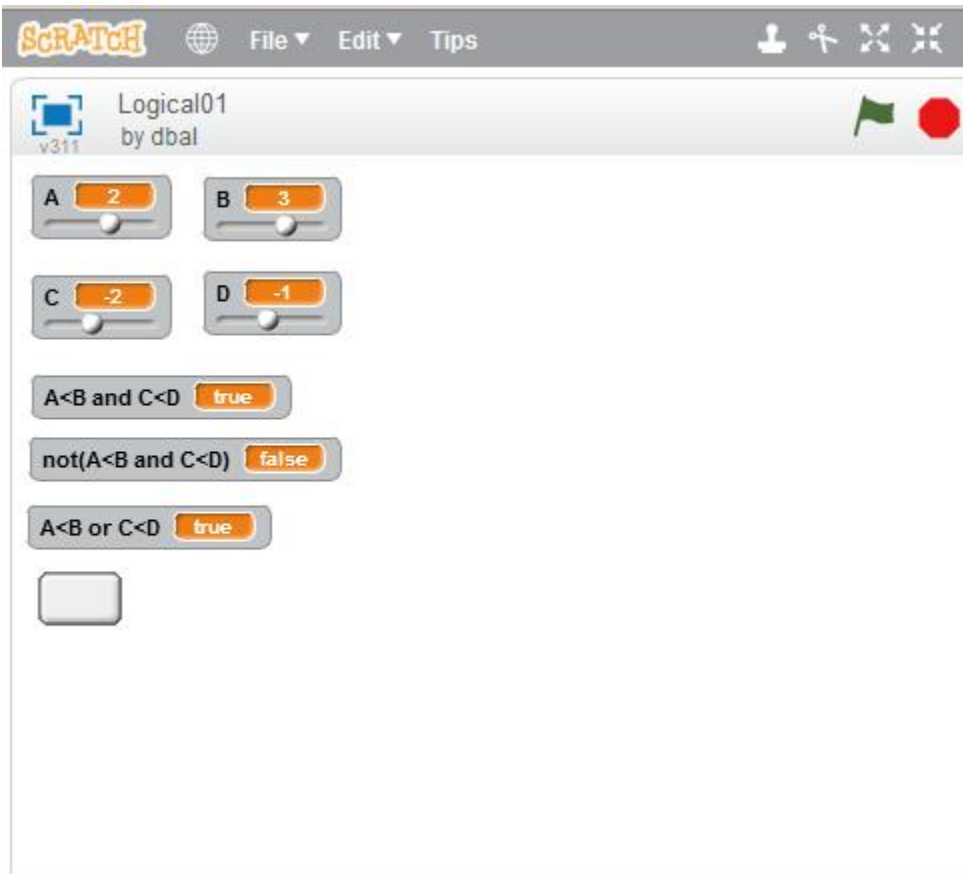


Image D. Program output for row 1 in Image C.

Image E. Program output for row 2 in Image C.

Figure

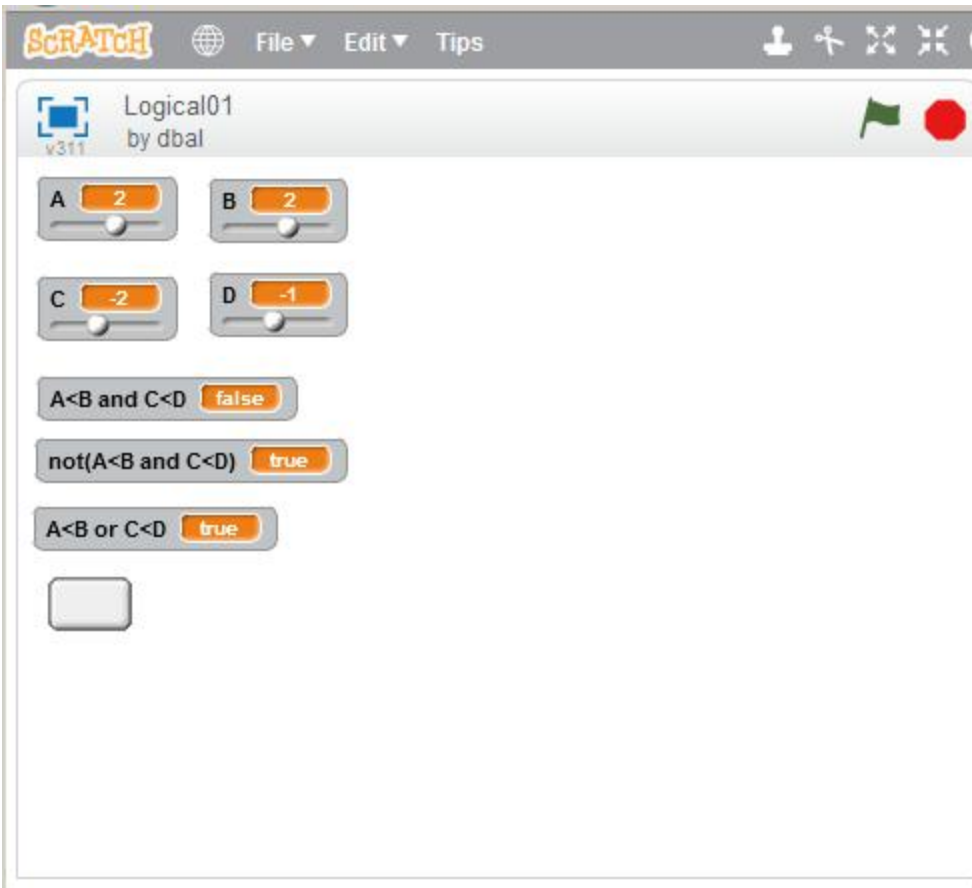


Image E. Program output for row 2 in Image C.

Image F. Program output for row 3 in Image C.

Figure

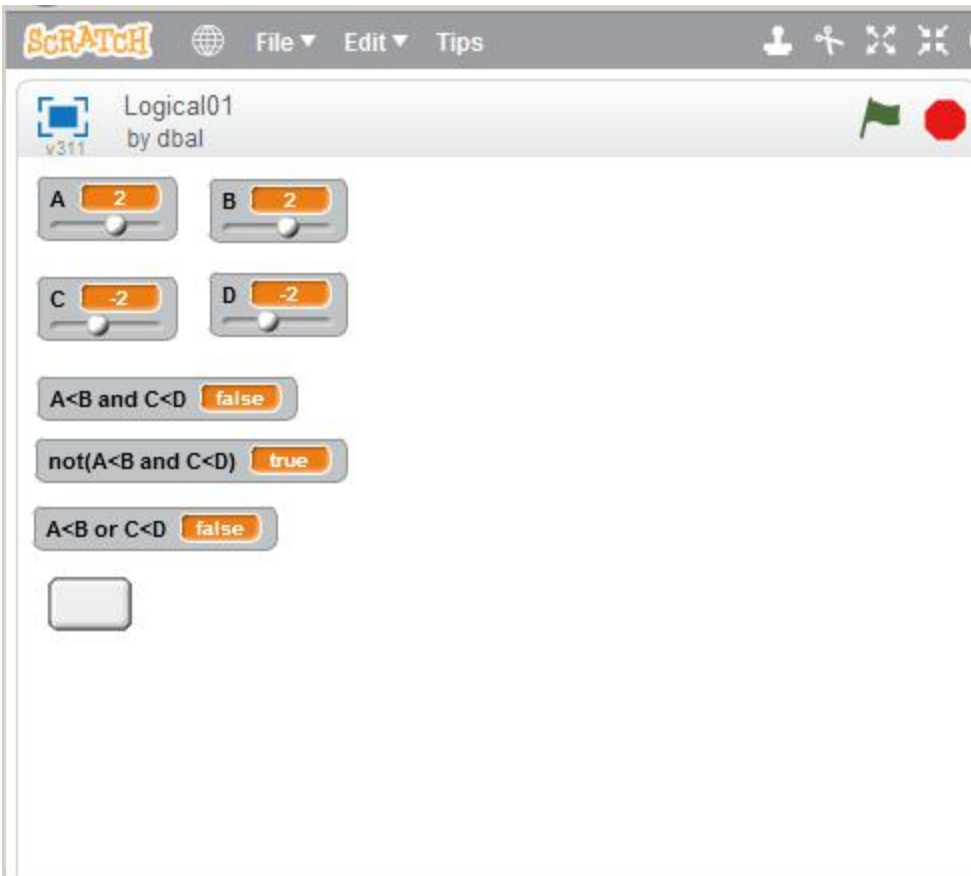


Image F. Program output for row 3 in Image C.

Discussion and sample code

[Image G](#) shows a full size screen shot of the programming panel from [Image A](#)

Image G. Programming panel for the program named Logical01.

Figure



Image G. Programming panel for the program named Logical01.

The top script in [Image G](#) initializes all of the variables to 0 when the user clicks the green flag. You already know about code like that from what you learned in earlier modules.

Note that the **not** operator in the third script is a unary operator because it has only one operand. It also uses the prefix format because it appears to the left of its operand. If you can't see that in [Image G](#), open the green **Operators** toolbox and take a look at the block for the **not** operator. It is more obvious there.

Compare Image G with Image C

If you compare the relational-logical expressions defined by the statements in the green areas of [Image G](#) with the expressions in the column headers in [Image C](#), you should see a strong resemblance.

Parentheses are used to eliminate ambiguity and to clearly identify the operands of each operator in [Image C](#). A 3D optical illusion is used for the same purpose in [Image G](#). For example, it appears that everything to the right of the **not** operator in [Image G](#) is a little closer to the viewer than the green platform on which the word **not** is printed.

No detailed explanation of the construction is required

In the past several modules, I have walked you through the detailed drag, drop, insert, and typing steps required to produce scripts similar to those shown in [Image G](#). Hopefully, you understand the mechanics of that process by now and it should no longer be necessary for me to walk you through the process.

Program behavior at runtime

The user begins by clicking the green flag in the upper right corner of the stage to initialize all of the variables shown in [Image D](#) to 0. Then the user can change the value in each of the variables **A** , **B** , **C** , and **D** by moving the sliders shown in [Image D](#).

Order of execution of scripts

Following that, when the user clicks the button at the bottom of [Image D](#), each of the bottom three scripts in [Image G](#) is executed. Note that they are not all executed simultaneously due to practical limitations in the construction of most computers. However, I can't tell you the exact order in which they are executed because, as far as I know, that is not public information. It might be logical to assume that the individual scripts are executed from top to bottom, but such an assumption may not be correct.

Not important in this program

The order in which the three scripts are executed doesn't really matter in this program because the end result will be the same regardless of the order of execution. However, if two different scripts were to make modifications to the value of the same variable, the order in which those modifications are made could matter a lot. I mention this simply as a word to the wise. Be careful and make certain that you don't write different event handlers that respond to the same event and modify the same data in different ways even if that seems to appeal to your organizational sensibilities. In that case, you should combine the different event handlers into a single event handler because the order of execution of the code within a script is well defined.

Evaluate the relational-logical expressions and take appropriate action

For each of the bottom three scripts in [Image G](#), if the relational-logical expression given by the green blocks with the embedded orange blocks evaluates to true, a value of true is set into and displayed by the variable identified by the statement immediately below the word **if** . Otherwise, a value of false is set into and displayed by that variable as shown in [Image D](#) through [Image F](#).

An online version of this program is available

A copy of this program has been posted online for your review (see [Resources](#) for the URL) . If you don't find the program using that URL, search the Scratch site for the user named dbal.

Run the program

I encourage you to use the information provided above to write this program. Experiment with the code, making changes, and observing the results of your changes. Make certain that you can explain why your changes behave as they do.

Just for fun, see if you can create a startup screen that displays text similar to the title screen or the credits screen at a movie.

I also encourage you to write the programs described below.

Student programming projects

Copies of these two programs have been posted online for your review (see [Resources](#) for the URL) . If you don't find the program using that URL, search the Scratch site for the user named dbal.

The program named Logical02

This Scratch program illustrates the use of relational and logical operators to determine if:

- The value of a variable named **LeftOperand** is *less than or equal to* the value of a variable named **RightOperand** .
- The value of the variable named **LeftOperand** is *greater than or equal to* the value of the variable named **RightOperand** .

The following four variables are created, displayed on the Stage, and set to zero when the user clicks the green flag:

1. **LeftOperand** (a slider variable)
2. **RightOperand** (a slider variable)
3. **LessThanOrEqual**
4. **GreaterThanOrEqual**

A button is also created and displayed on the Stage as shown in [Image H](#). Image H. Program output from the program named Logical02.

Figure

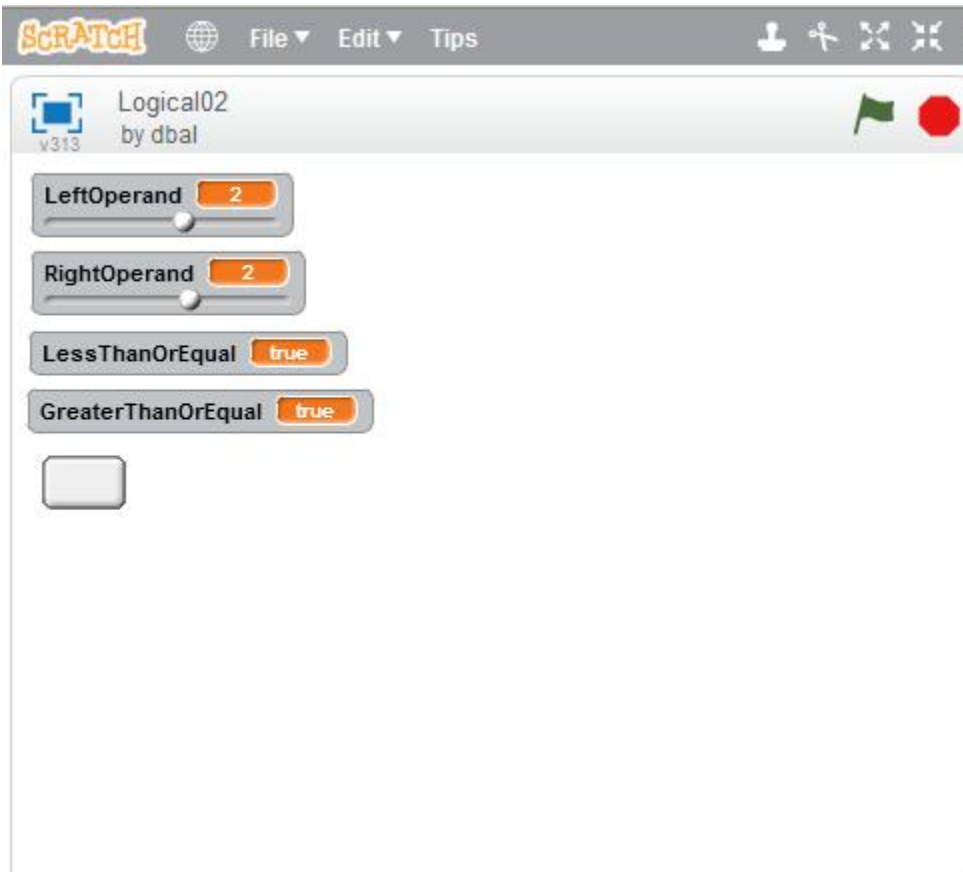


Image H. Program output from the program named Logical02.

In addition, variables named **Equals** , **GreaterThan** , and **LessThan** are also created but not initialized or displayed. These variables are used by the code to achieve the desired result when the user clicks the button.

The value of each slider variable is set by the user moving the thumb on the slider.

When the user clicks the button, if the left operand is *less than or equal to* the right operand, the third variable in [Image H](#) is set to true. Otherwise, that variable is set to false.

If the left operand is *greater than or equal to* the right operand, the fourth variable in [Image H](#) is set to true. Otherwise it is set to false.

*Note, two of the scripts in my version of the program use variables named **Equals** , **GreaterThan** , and **LessThan** whose values are set by the three other scripts. Because I'm unsure as to the order of execution of the scripts, I inserted a delay of 0.1 second at the beginning of each of those two scripts in an attempt to ensure that the execution of the other three scripts will be completed before an attempt is made to use the values of those variables. This is a crude way to deal with this issue, and is not recommended for anything other than demonstration code.*

The program named Logical03

This Scratch program is a modification of the program named Logical02. This is a relatively challenging program to write and requires an understanding of DeMorgan's theorem (see [Resources](#)) .

The behavior of this program is identical to the behavior of the program named **Logical02** . However, it is likely that you used the **or** operator to write that program. This program does not use an **or** operator.

Instead, this program uses the **not** operator in conjunction with the **and** operator along with DeMorgan's theorem to produce behavior that is identical to the program named **Logical02** .

Summary

I provided an explanation of logical operators and truth tables. Then I presented and explained a Scratch program that illustrates the use of the logical **and** , **or** , and **not** operators when used in conjunction with relational operators.

Finally, I provided the specifications for two student-programming projects for you to demonstrate your understanding of what you learned from the first program and from earlier modules. One of the student-programming projects is relatively difficult requiring an understanding of DeMorgan's theorem (see [Resources](#)) .

Copies of all three programs have been posted online for your review (see [Resources](#) for the URL) . If you don't find the program using that URL, search the Scratch site for the user named dbal.

What's next?

Future modules using the Scratch programming language will continue to deal with *selection* and will also deal with *loops* .

Resources

General resources

- [Scratch home](#)
- [Scratch tutorials](#)
- [Scratch explanatory video](#)
- [Scratch tutorials from MIT](#)
- [Scratch tutorial - Digital Art for All](#)
- [Scratch for Budding Computer Scientists](#)
- [Learn Scratch at learnscratch.org](#)
- [Scratch Tutorial - Space Shuttle Mission STS-2020](#)
- [SCRATCH TUTORIALS ICT In Primary Education 2012-2013](#)
- [Scratch Wiki](#)
- [Scratch WikiTable of ContentsWebsite - Scratch Wiki](#)
- [Scratch WikiTable of ContentsTutorials - Scratch Wiki](#)
- [Scratch WikiTable of ContentsProgram - Scratch Wiki](#)
- [Tutorial about Variables - Scratch Wiki](#)
- [Scratch support MIT web site](#)
- [Scratch resources at Scratch.ie](#)
- [ScratchEd Resources](#)

- [Scratch project Day Dream](#)
- [Scratch project Son of String Art](#)
- [Scratch Project Scratch Tutorial](#)
- [DeMorgan's theorem](#)

Programs used in this collection

- [Variable01](#) - Online version of program
- [Variable02](#) - Online version of student programming project
- [Variable03](#) - Online version of student programming project
- [IfSimple01](#) - Online version of program
- [IfWithVar01](#) - Online version of student programming project
- [Arithmetic01](#) - Online version of program
- [Arithmetic02](#) - Online version of student programming project
- [Relational01](#) - Online version of program
- [Relational02](#) - Online version of student programming project
- [Logical01](#) - Online version of program
- [Logical02](#) - Online version of student programming project
- [Logical03](#) - Online version of student programming project

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Scr0360: Logical Operators in Scratch 2.0
- File: Scr0360.htm
- Published: 05/16/13

Note: Disclaimers:

Financial : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.

I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module. In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.

Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Scr0370: The repeat loop in Scratch 2.0

In this module, you will Learn the difference between a definite loop and an indefinite loop. I will explain two Scratch programs that approximate the Scratch equivalent of a for loop in other programming languages. In Scratch, it is called a repeat loop. I will provide a brief introduction to the topic of animation. I will also explain the concept of costumes in Scratch. The programs explained in this module will illustrate the use of costumes to implement a crude animation that makes it appear that a cat sprite is walking.

Table of Contents

- [Preface](#)
 - [Viewing tip](#)
 - [Images](#)
- [General background information](#)
 - [Definite versus indefinite loops](#)
 - [A definite loop example](#)
 - [Explanation of the definite loop](#)
 - [An indefinite loop example](#)
 - [Explanation of the indefinite loop](#)
 - [The while and for keywords](#)
- [Preview](#)
- [Discussion and sample code](#)
 - [The program named ForLoop01](#)
 - [Why set count to 0?](#)
 - [Purpose of the pseudocode](#)
 - [What's new in this Scratch program?](#)
 - [Animation](#)

- [Costumes](#)
- [The cat sprite has two costumes](#)
- [Loop blocks](#)
- [The repeat \(numeric value\) block](#)
- [Setting the number of repeats \(iterations\)](#)
- [Can also use a variable](#)
- [The wait block](#)
- [The switch costume block](#)
- [The program code](#)
- [Behavior of the program](#)
- [The repeat block](#)
- [Run the online version](#)
- [The program named ForLoop02](#)
 - [The program code](#)
 - [Create a variable](#)
 - [The initialization code](#)
 - [The repeat block](#)
 - [The operational difference](#)
- [Online versions of these programs are available](#)
- [Run the programs](#)
- [Student programming project](#)
- [Summary](#)
- [What's next?](#)
- [Resources](#)
 - [General resources](#)
 - [Programs used in this series](#)
- [Miscellaneous](#)

Preface

[Scratch 2.0](#) (released May 9, 2013) is the second major version of Scratch to be released during the life of the product. Among other things, it features

a redesigned editor and website, and allows you to edit projects directly from your web browser.

This module (*tutorial*) is one in a collection of modules designed to help beginners of all ages (*8 and up*) learn how to create the code for computer programs using [Scratch 2.0](#). Information is provided not only for the beginners themselves but also for their parents and teachers where appropriate.

In this module, you will Learn the difference between a *definite loop* and an *indefinite loop*. I will explain two Scratch programs that approximate the Scratch equivalent of a **for** loop in other programming languages. In Scratch, it is called a **repeat** loop. One of the programs causes a block of code to be executed a fixed number of times when the user clicks the green flag. The other program causes a block of code to be executed a variable number of times when the user clicks a sprite.

I will provide a brief introduction to the topic of animation. I will also explain the concept of *costumes* in Scratch. Both of the programs mentioned above will illustrate the use of costumes to implement a crude animation that makes it appear that a cat sprite is walking.

Viewing tip

I recommend that you open another copy of this module in a separate browser window and use the following links to easily find and view the images while you are reading about them.

Images

- [Image A](#). A definite loop for eating cookies.
- [Image B](#). An indefinite loop for eating cookies.
- [Image C](#). Pseudocode for the program named ForLoop01.
- [Image D](#). The costumes tab for a new Scratch project.
- [Image E](#). Various Scratch blocks.

- [Image F](#). Program code for the program named ForLoop01.
- [Image G](#). The cat on the Stage.
- [Image H](#). Program ForLoop02 output after clicking the green flag.
- [Image I](#). Program ForLoop02 output after clicking the cat.
- [Image J](#). Program code for the program named ForLoop02.
- [Image K](#). ForLoop03 output after clicking the green flag.

General background information

In an earlier module (see [Resources](#)) , I told you that any programming logic problem could be solved using an appropriate combination of only three programming structures, none of which are complicated. The three structures are known generally as:

- The sequence structure.
- The selection or decision structure.
- The loop, repetition, or iteration structure.

I explained the sequence and selection structures in earlier modules. I will concentrate on the loop structure in this and the next module.

Definite versus indefinite loops

Loop structures in programming fall into two broad categories:

- Definite loops
- Indefinite loops

There are numerous sub-categories within these broad categories. I will illustrate the difference between a *definite* loop and an *indefinite* loop with two real-world examples.

A *definite* loop example

Assume that you are confronted with a full box of cookies. Assume also that you have more self-control than most of us, and that you decide you can eat three cookies, but no more than three, in order to keep your waistline under control. You might eat the three cookies using an algorithm something like that shown in [Image A](#).

Image A. A definite loop for eating cookies.

Figure

```
for count = 1 to 3
  Take cookie from box
  Eat cookie
  Increase count by one
  Go back to the test at the top of the loop
Stop eating cookies
```

Image A. A definite loop for eating cookies.

Explanation of the definite loop

Being the self-controlled individual that you are, you would set your cookie limit to 3 and you would set **count** to 1. You would then test **count** to see if it is within the range from 1 to 3 inclusive. If so, you would take a cookie from the box and eat it. Then you would increase the value of **count** by 1 and go back to the top of the loop.

Back at the top of the loop, you would once again test the value of **count** to determine if it is still within the range from 1 to 3 inclusive. If so, you would repeat the process, getting and eating another cookie, increasing the value of **count** by 1, and going back to the top of the loop.

When you find that the value of **count** has advanced to 4, you would recognize that this is outside the range of 1 to 3 inclusive. As a result, you would terminate the loop and stop eating cookies. Well done!

An indefinite loop example

Assume once again that you are confronted with the same full box of cookies. However, like many of us, you aren't blessed with a lot of self-control. In that case, you might eat cookies using the algorithm shown in [Image B](#).

Image B. An indefinite loop for eating cookies.

Figure

Set the value of `stillHungry` to true

```
while ((stillHungry is true) and (box is not empty))
```

```
    Take cookie from box
```

```
    Eat cookie
```

```
    If I am no longer hungry
```

```
        Set stillHungry to false
```

```
    Go back to the test at the top of the loop
```

```
Stop eating cookies
```

Image B. An indefinite loop for eating cookies.

Explanation of the indefinite loop

Unfortunately, for many of us, as soon as we see the full box of cookies, our **stillHungry** variable gets set to true.

We begin the process by performing a test to determine if **stillHungry** is true *and* the box of cookies is not empty. If that test returns true, we take a cookie from the box and eat it. Then we perform another test. If we are no longer hungry at that point, we set our **stillHungry** variable to false and go back to test at the top of the loop. Otherwise, we allow our **stillHungry** variable to remain true.

Back at the top of the loop, we test again to determine if **stillHungry** is true, and the box of cookies is not empty. If the test returns true, we go through the process again, eating another cookie, etc.

Eventually, either our **stillHungry** variable will be set to false, or the cookie box will become empty. At that point, the test will return false. We will terminate the loop and will stop eating cookies even though we might still be hungry for more cookies.

The *while* and *for* keywords

Most modern programming languages use the keywords **while** and **for** to implement loop structures such as this and you will see those keywords in use once you get to Java, C++, C#, etc.

However, Scratch does not use those keywords to implement loops. Regardless, when I present pseudocode examples to explain Scratch programs, I will continue to use **while** and **for** to get you accustomed to that terminology.

Preview

In this module, I will present and explain the following two Scratch programs:

- **ForLoop01** - This program illustrates the Scratch equivalent of a **for** loop (*or something similar to a **for** loop*) in other languages. In Scratch, it is called a **repeat** loop.
- **ForLoop02** - This program also illustrates the Scratch equivalent of a **for** loop in other languages. However, this program takes a different approach to specifying the number of times that the code in the loop is executed.

In addition, I will provide the specifications for a student-programming project for you to demonstrate your understanding of what you learned from the two programs listed above. In addition, this programming project will

require you to do some independent research into the manner in which Scratch sprites can communicate with one another.

Discussion and sample code

The program named ForLoop01

This program illustrates the Scratch equivalent of a **for** loop (*or something similar to a **for** loop*) in other programming languages. In Scratch, it is called a **repeat** .

Why do I keep saying "something similar to a for loop?" The reason is that a for loop in other programming languages has many facets. The repeat loop in Scratch is much more restricted than a for loop in those other languages.

This program places a cat sprite having two costumes on the stage. When the user clicks the green flag, code is executed that is equivalent to the pseudocode shown in [Image C](#).

Image C. Pseudocode for the program named ForLoop01.

Figure

Initialize cat position, orientation, and costume

```
for(count = 0;count < 24,count = count + 1){  
    Cat moves ten steps forward.  
    Cat waits 0.1 second  
    Cat switches to costume #2  
    Cat turns 15 degrees  
    Cat waits 0.1 second  
    Cat switches back to costume #1  
}//end for loop
```

Image C. Pseudocode for the program named ForLoop01.

Why set count to 0 ?

Most modern programming languages typically start counting with 0 instead of 1, but that is not necessarily the case with Scratch. You will learn more about that when you get into Java, C++, C#, etc.

Purpose of the pseudocode

My purpose in providing the pseudocode is to help you better understand the Scratch code that follows. I am also preparing you for the use of other programming languages later. I recommend that you compare the pseudocode with the Scratch code, and compare both of them with what you see when you run the program.

What's new in this Scratch program?

This program contains several new code blocks that haven't been used in previous modules:

- The **switch costume** block
- The **repeat** block
- The **wait** block

The only one of these blocks that I would put in the *programming fundamentals* category is the **repeat** block, and even the syntax of that block is quite a bit different from other modern programming languages. However, the other blocks are some of the things that cause Scratch programming to be fun, so I will use and explain such blocks as we go along. My only caution is that you should not expect to find those

commands in other programming languages, at least not using the same syntax.

Animation

Since the earliest days of animation, two-dimensional (2D) animation has been created by presenting a series of drawings to the viewer in rapid succession where each drawing was similar to but slightly different from the previous drawing. Because of the property of human vision to blur one image into the next, if the images are sufficiently similar and they are presented in rapid succession, the human mind perceives the series of images as continuous motion. This is how motion pictures work, and it is also how the cartoons that you see on television work.

Costumes

Animation can be created in a couple of different ways in Scratch. One way is through the use of something called **costumes**. For example, when you create a new project in Scratch, an image of a cat having two costumes is automatically added to the project. [Image D](#) shows a screen shot of the center panel in the user interface for a new project with the **Costumes** tab selected.

Image D. The costumes tab for a new Scratch project.

Figure

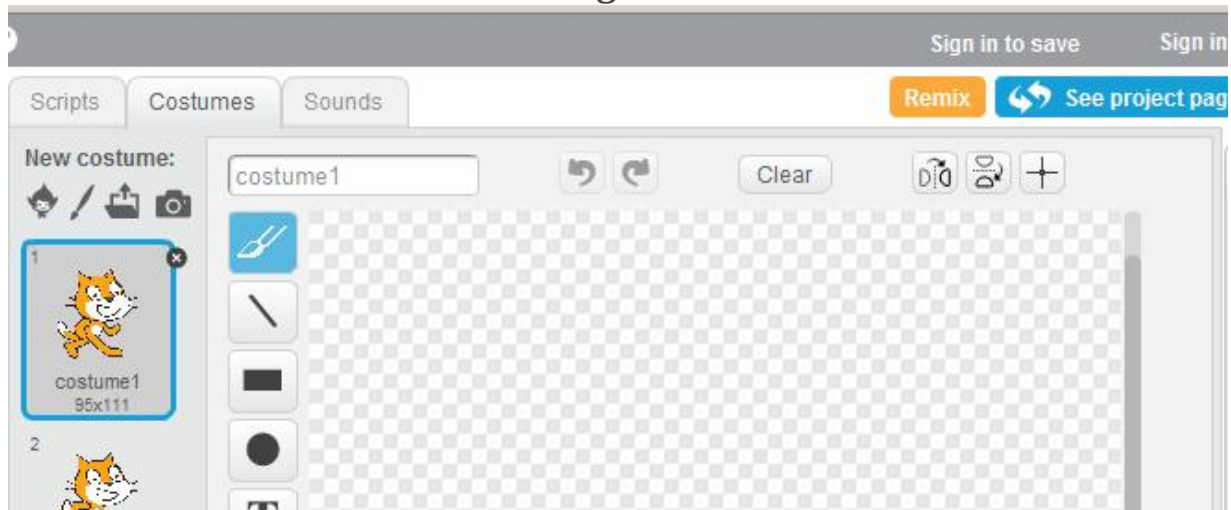




Image D. The costumes tab for a new Scratch project.

[Image D](#) shows a lot more than just the current costumes. The current costumes are shown in the upper-left corner of [Image D](#).

The cat sprite has two costumes

As you can see, the cat sprite has two costumes, which are really two different images of the cat. The images are designed to make it look like the cat is walking if the two images are repeated in rapid succession. You can also see that each costume has a name, with the default names in this case being:

- costume1
- costume2

If you choose to do so, you can click in the boxes containing the names and modify them. Also, if you choose to do so, you can use the built-in drawing program to modify the drawing. However, most of that is beyond the scope of this module so we won't get into it here.

With only two costumes for the cat, the walking motion is not very smooth. Somewhere on the Scratch website, I saw a project that had sixteen different costumes for the cat. When those sixteen different costumes are displayed in succession, the cat appears to walk very smoothly.

Loop blocks

If you click the tan **Control** button in the Scratch user interface, you will expose a large number of blocks that are generally used to control the flow of the program. The following control blocks that are exposed by that button can be used to construct loops:

Note:

repeat (numeric value) - (see

The repeat (numeric value) block

The **repeat (numeric value)** block (*referred to hereafter simply as the repeat block*) provides the mechanism by which you can cause a set of actions to be executed a specified number of times. (See [Image E.](#))

You create a **repeat** block that will do something useful by inserting programming blocks into the mouth of a **repeat** block. When you drag a raw **repeat** block into the center panel, it looks like the tan image shown in [Image E.](#)

Image E. Various Scratch blocks.

Figure

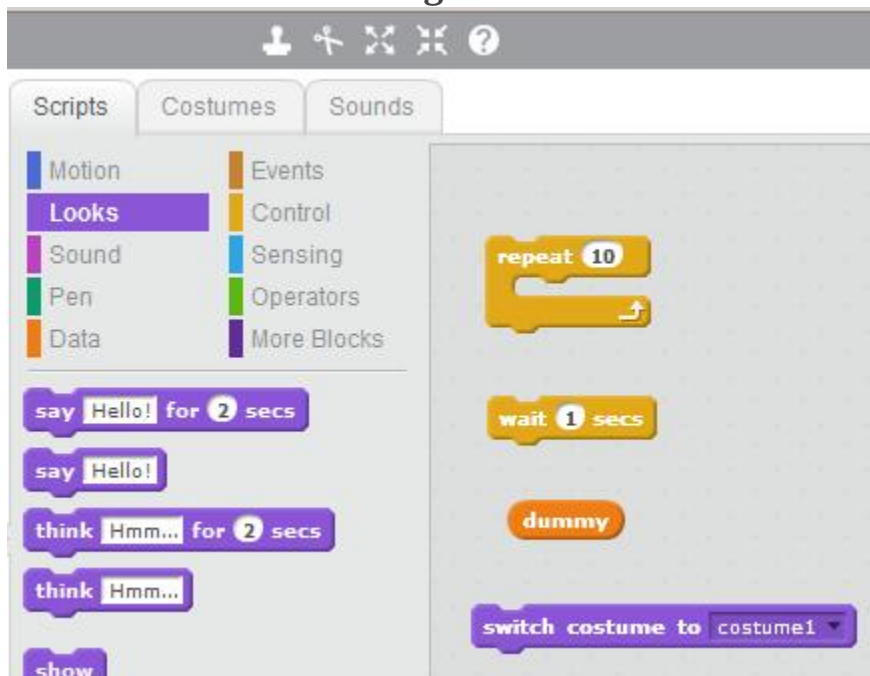


Image E. Various Scratch blocks.

Setting the number of repeats (iterations)

By default, the white box in the **repeat** block contains a literal numeric value. You can modify that value and specify the number of times that the

code in the **repeat** block will be executed. When you use a literal value, the code will be executed the same number of times every time you run the program.

Can also use a variable

Note that I also created a dummy variable in [Image E](#) for illustration purposes. Because the shape of a variable block is the same as the shape of the white box in the **repeat** block, you can also drop a variable block into the box. When you do this, the value of the variable will be used to determine how many times the code in the block will be executed. That number may be different each time the **repeat** block is executed.

The wait block

[Image E](#) also shows a **wait** block, which can be dragged into the center panel from the **Control** toolbox. Like the **repeat** block, the **wait** block has a white box into which you can either enter a literal value or drop a variable block.

The program will pause for a specified number of seconds (*which may be a decimal fraction*) each time the **wait** block is executed. If you enter a literal value into the box, the program will pause for the same amount of time each time the **wait** block is executed. If you drop a variable block into the box, the length of the pause will be the current value stored in the variable, which may be different each time the block is executed.

The switch costume block

Finally, [Image E](#) shows a purple **switch to costume** block. This block is available in the purple **Looks** toolbox when a sprite icon is selected immediately below the stage. However, when the Stage icon is selected, the block changes to one labeled **switch to backdrop** . (*A discussion of backdrops will be deferred to a future module.*)

As you can see, the **switch to costume** block has a pull-down list that allows you to select a specific costume for the block when you write the program. The list contains all of the costumes belonging to the selected sprite. In the case of the default cat that appears for new projects, there are two costumes in the pull-down list because the cat has two costumes as shown in [Image D](#).

The program code

The code for this program is shown in [Image F](#).
Image F. Program code for the program named ForLoop01.

Figure

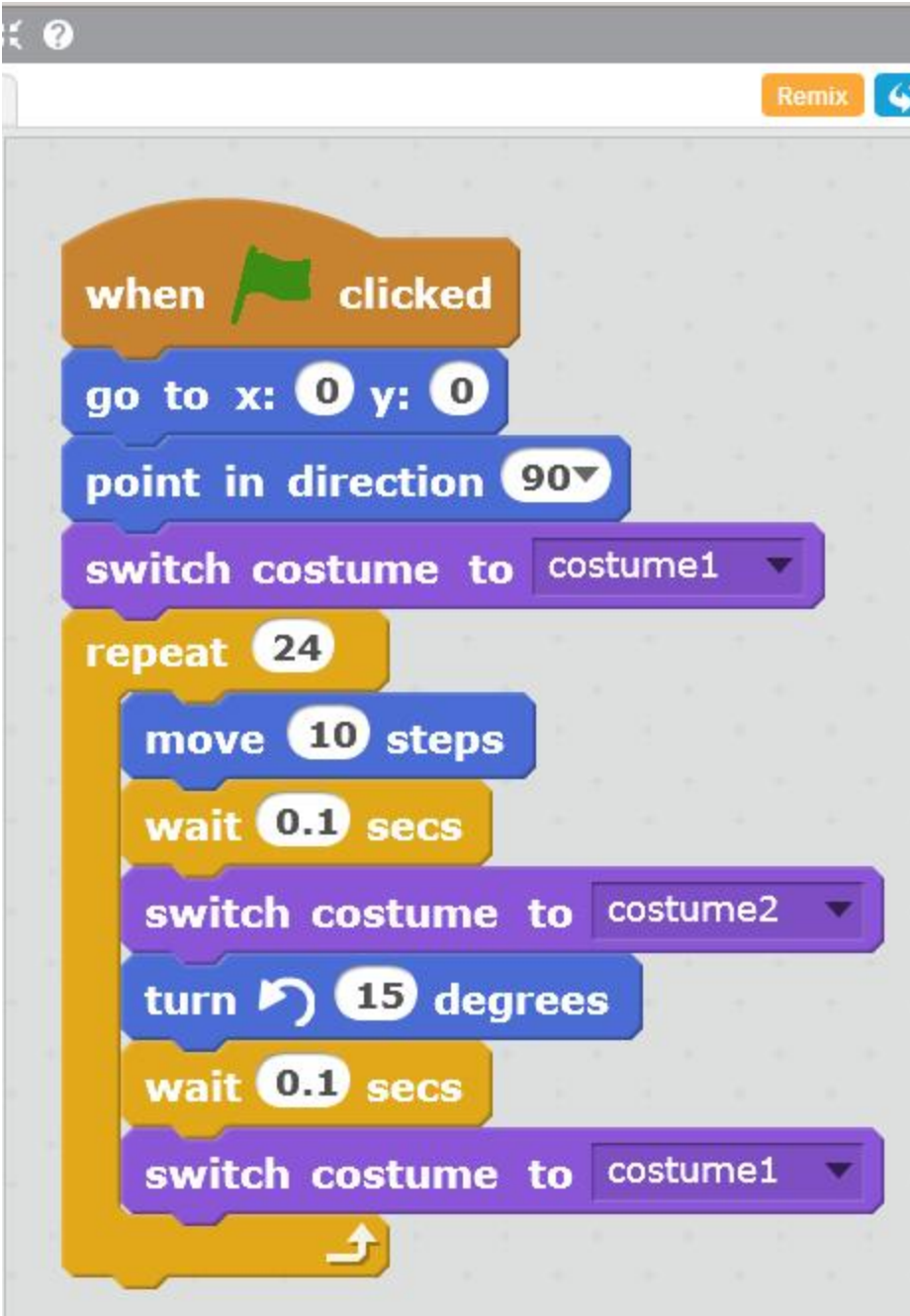


Image F. Program code for the program named ForLoop01.

Behavior of the program

When the user clicks the green flag, the cat moves to the center of the stage (*the origin of the Cartesian coordinate system*) and turns to face the viewer's right. In addition, the cat switches to costume1 (see [Image D](#)) even if that isn't necessary at this point. (*Except for switching costumes, you have seen code like this in earlier modules.*)

The repeat block

Then the code inside the **repeat** block shown in [Image F](#) is executed 24 times in succession.

During each execution (*iteration*) of the **repeat** block, the cat:

- Moves 10 steps forward.
- Waits 0.1 second.
- Switches to costume2.
- Rotates around its own origin by 15 degrees.
- Waits an additional 0.1 seconds.
- Switches back to costume 1.

This causes the cat to appear to walk (*in a rather jerky fashion*) in a circle (see [Image G](#)), ending up very close to where it started.

Image G. The cat on the Stage.

Figure



Image G. The cat on the Stage.

Run the online version

In order to get the full impact of this program, you will either need to create the program yourself or run the online version (see [Resources](#)) .

The program named ForLoop02

This program is an upgrade of the program named **ForLoop01** . The behavior of this program is the same as the earlier program with the following exceptions:

- A slider variable named **numberLoops** is placed in the upper left corner of the Stage as shown in [Image H](#).
- When the user clicks the green flag, orthogonal axes are drawn through the origin of the Stage and the cat is placed at the origin.

- When the user clicks the cat, the number of times the code in the **repeat** block is executed is determined by the current value of the slider variable.
- As the code in the **repeat** block is repetitively executed, the path of the cat is traced out on the Stage as a series of short straight line segments, producing the circular path shown in [Image I](#).

Image H. Program ForLoop02 output after clicking the green flag.

Figure

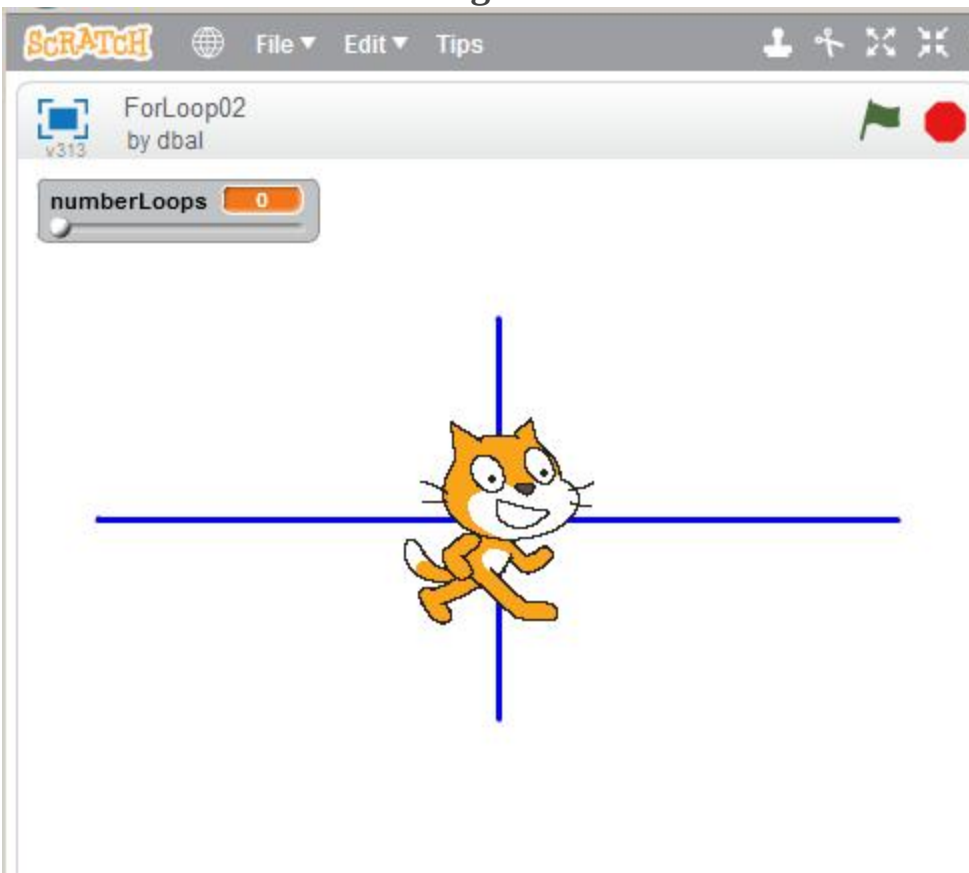


Image H. Program ForLoop02 output after clicking the green flag.

Image I. Program ForLoop02 output after clicking the cat.

Figure

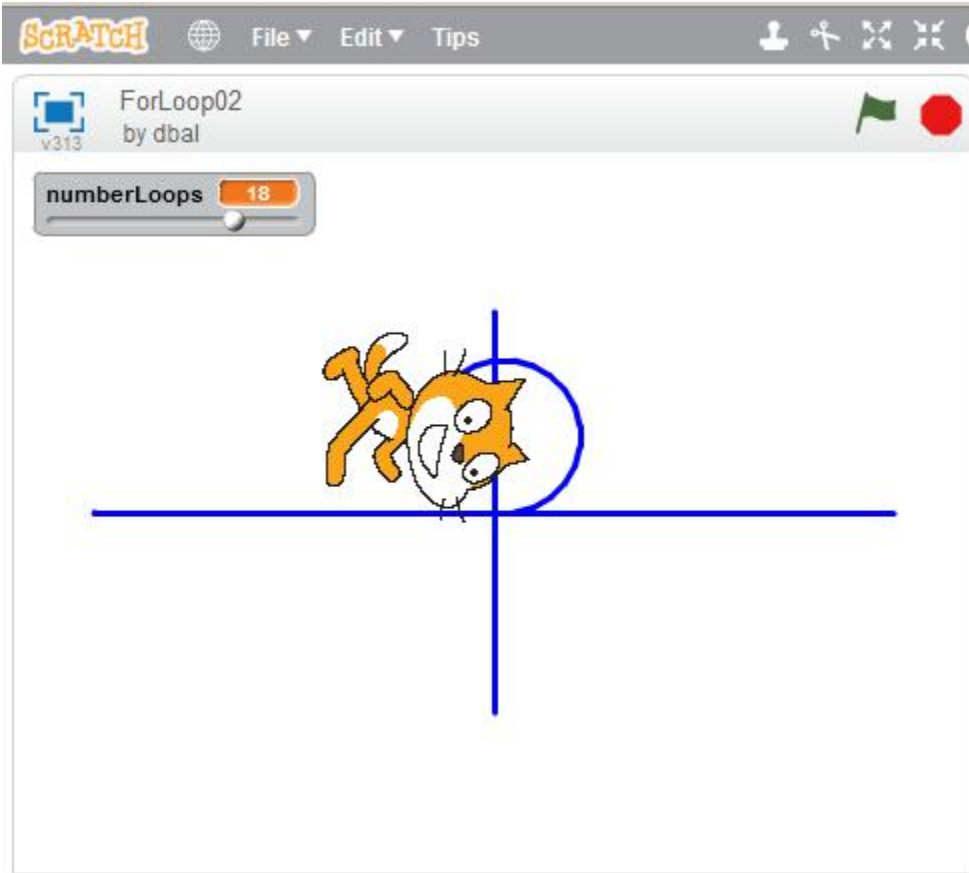


Image I. Program ForLoop02 output after clicking the cat.

The program code

The code for this program is shown in [Image J](#).

Image J. Program code for the program named ForLoop02.

Figure

Icons:

Scripts | Costumes | Sounds

Motion | Looks | Sound | Pen | **Data** | Events | Control | Sensing | Operators | More Blocks

Make a Variable

☒ **numberLoops**

set numberLoops to 0

change numberLoops by 1

show variable numberLoops

hide variable numberLoops

Make a List

when clicked

clear

pen up

set pen size to 3

go to x: -200 y: 0

pen down

go to x: 200 y: 0

pen up

go to x: 0 y: 100

pen down

go to x: 0 y: -100

pen up

go to x: 0 y: 0

point in direction 90

switch costume to costume1

set numberLoops to 0

pen down

when this sprite clicked

repeat numberLoops

move 10 steps

wait 0.1 secs

switch costume to costume2

turn 15 degrees

wait 0.1 secs

switch costume to costume1

Image J. Program code for the program named ForLoop02.

Create a variable

The code in the left panel in [Image J](#) creates a variable with a slider named **numberLoops** and causes it to be displayed on the Stage as shown in [Image I](#).

The initialization code

Although there is quite a lot of code that is executed when the green flag is clicked, there is nothing new there. All of that code has been explained in this or earlier modules. For review, the green *pen* blocks and the blue *go to* blocks are used together to draw the Cartesian coordinates. Note that the pen is down when the initialization code finishes executing. As a result, a line will be drawn whenever the cat sprite moves.

The repeat block

The only thing that is really new to this program is the use of the variable block named **numberLoops** to specify the number of times that the code in the **repeat** block will be executed each time the cat is clicked. Otherwise the code in the **repeat** block in [Image J](#) is the same as the code in the **repeat** block shown in [Image F](#).

The operational difference

The operational difference is that each time the repeat block in [Image F](#) is executed, the code inside the **repeat** block will be executed 24 times. Each time the **repeat** block in [Image J](#) is executed, the number of times that the code inside the repeat block will be executed is specified by the current position of the slider in [Image I](#). This, in effect converts the static program named **ForLoop01** into an interactive program in which the user has some control over the program behavior at runtime.

As mentioned earlier, because the pen is down, a line is drawn each time the cat sprite moves. That is what draws the circular path shown in [Image I](#).

Online versions of these programs are available

Copies of these two programs have been posted online for your review (*see [Resources](#) for the URL*). If you don't find the programs using that URL, search the Scratch site for the user named dbal.

Run the programs

I encourage you to use the information provided above to write and run these two programs. Experiment with the code, making changes, and observing the results of your changes. Make certain that you can explain why your changes behave as they do.

Just for fun, see if you can add a drumbeat that keeps time as the cat walks.

I also encourage you to write the program described below.

Student programming project

Write a Scratch program named **ForLoop03**. This program places a cat sprite, a slider variable, and a button on the stage as shown in [Image K](#). The cat sprite has two costumes.

Image K. ForLoop03 output after clicking the green flag.

Figure

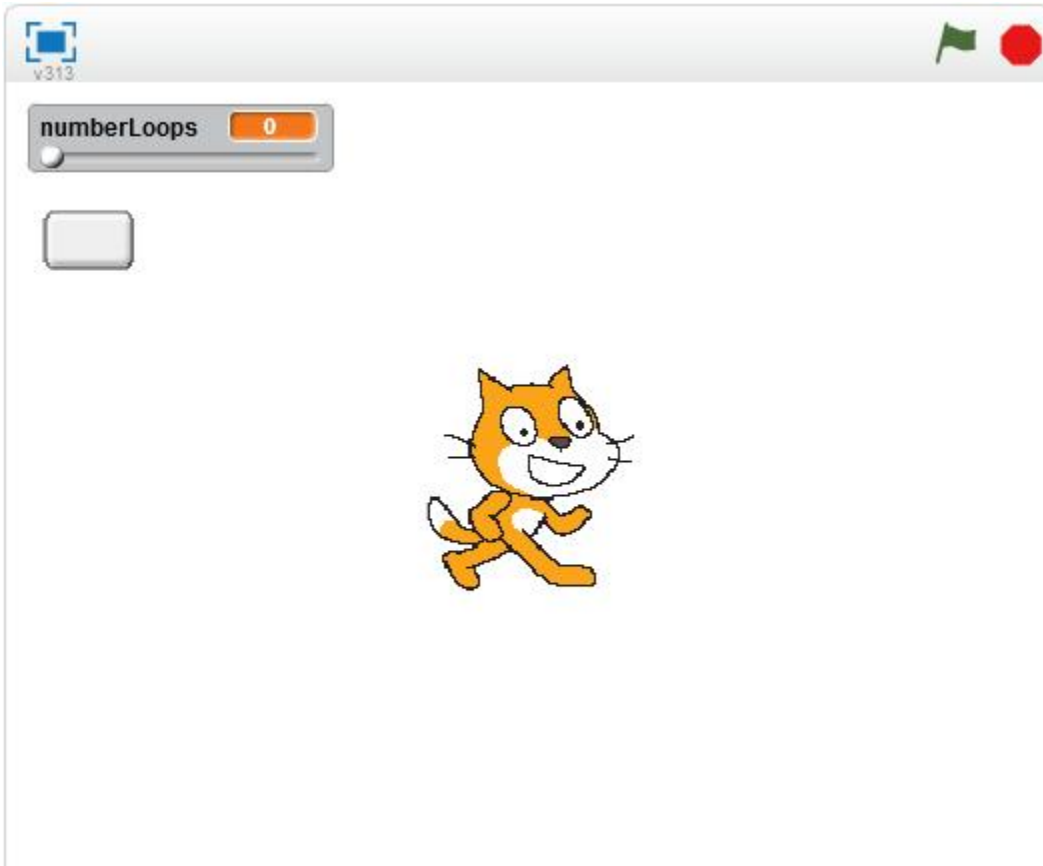


Image K. ForLoop03 output after clicking the green flag.

This program is an upgrade of the program named **ForLoop02** , but is more complex than that program. In particular, this program uses the ability of one sprite (*the button*) to communicate with another sprite (*the cat*) by broadcasting and receiving messages.

You may have to do some online research to learn how to fire and handle broadcast events. Hint: See the bottom three blocks in the Events toolbox.

When the user clicks the button, this program behaves just like the program named **ForLoop02** behaves when the user clicks the cat in that program. *(Note, however, that the drawing of axes and lines was omitted from this program for simplicity.)*

As in the previous program, this program executes its animation using a **repeat** block. Also as in the previous program, the number of times that the code in the **repeat** block is executed is specified by the current position of the slider.

Online version is available

A copy of this program has been posted online for your review (see [Resources](#) for the URL) . If you don't find the program using that URL, search the Scratch site for the user named dbal.

Summary

I began by providing a real-world example of the difference between a *definite* loop and an *indefinite* loop.

Then I presented and explained two Scratch programs that illustrate the Scratch equivalent of *(or something similar to)* a **for** loop in other programming languages. In Scratch, it is called a **repeat** loop. One of the programs caused a block of code to be executed a fixed number of times when the user clicked the green flag. The other program caused a block of code to be executed a variable number of times when the user clicked a sprite.

I provided a brief introduction to the topic of animation and I explained the concept of *costumes* in Scratch. Both programs illustrate the use of costumes to implement crude animation that makes it appear that a cat sprite is walking.

I also provided a student programming project.

What's next?

The next module will concentrate on other variations of Scratch loops.

Resources

General resources

- [Scratch home](#)
- [Scratch tutorials](#)
- [Scratch explanatory video](#)
- [Scratch tutorials from MIT](#)
- [Scratch tutorial - Digital Art for All](#)
- [Scratch for Budding Computer Scientists](#)
- [Learn Scratch at learnscratch.org](#)
- [Scratch Tutorial - Space Shuttle Mission STS-2020](#)
- [SCRATCH TUTORIALS ICT In Primary Education 2012-2013](#)
- [Scratch Wiki](#)
- [Scratch WikiTable of ContentsWebsite - Scratch Wiki](#)
- [Scratch WikiTable of ContentsTutorials - Scratch Wiki](#)
- [Scratch WikiTable of ContentsProgram - Scratch Wiki](#)
- [Tutorial about Variables - Scratch Wiki](#)
- [Scratch support MIT web site](#)
- [Scratch resources at Scratch.ie](#)
- [ScratchEd Resources](#)
- [Scratch project Day Dream](#)
- [Scratch project Son of String Art](#)
- [Scratch Project Scratch Tutorial](#)
- [DeMorgan's theorem](#)

Programs used in this series

- [Variable01](#) - Online version of program
- [Variable02](#) - Online version of student programming project
- [Variable03](#) - Online version of student programming project

- [IfSimple01](#) - Online version of program
- [IfWithVar01](#) - Online version of student programming project
- [Arithmetic01](#) - Online version of program
- [Arithmetic02](#) - Online version of student programming project
- [Relational01](#) - Online version of program
- [Relational02](#) - Online version of student programming project
- [Logical01](#) - Online version of program
- [Logical02](#) - Online version of student programming project
- [Logical03](#) - Online version of student programming project
- [ForLoop01](#) - Online version of program
- [ForLoop02](#) - Online version of program
- [ForLoop03](#) - Online version of student programming project

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Scr0370: The repeat loop in Scratch 2.0
- File: Scr0370.htm
- Published: 05/18/13

Note: Disclaimers:

Financial : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.

I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module.

In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.

Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Scr0380: The forever and repeat until loops in Scratch 2.0

The purpose of this module is to teach you how to use the forever block and the repeat until block to create loop structures in Scratch 2.0.

Table of Contents

- [Preface](#)
 - [Viewing tip](#)
 - [Images](#)
- [General background information](#)
- [Preview](#)
- [Discussion and sample code](#)
 - [The program named ForeverLoop01](#)
 - [Program code for the Scratch v1.4 version of the program named ForeverLoop01](#)
 - [Program code for the Scratch v2.0 version of the program named ForeverLoop01](#)
 - [The last block in the script](#)
 - [The next costume block](#)
 - [The forever block](#)
 - [An if block](#)
 - [When the condition is true](#)
 - [No screen shot of the output](#)
 - [The program named ForeverLoop02](#)
 - [Comparison between repeat and forever/if blocks](#)
 - [No drop-in required for a repeat block](#)
 - [A drop-in is required for the if block](#)
 - [The forever block is more versatile than the repeat block](#)
 - [Can't connect to the bottom](#)
 - [Program code for the program named ForeverLoop02](#)
 - [Pseudocode for the program named ForeverLoop02](#)

- [A counter variable](#)
- [No screen shot of the output](#)
- [The program named RepeatUntil01](#)
 - [Program behavior](#)
 - [Program code for the program named RepeatUntil01](#)
 - [What does the cat do?](#)
 - [Doesn't have to be the last block in the script](#)
 - [Available online](#)
- [The loop blocks](#)
- [Run the programs](#)
- [Student-programming.project](#)
- [Summary](#)
- [What's next?](#)
- [Resources](#)
 - [General resources](#)
 - [Programs used in this series](#)
- [Miscellaneous](#)

Preface

[Scratch 2.0](#) (*released May 9, 2013*) is the second major version of Scratch to be released during the life of the product. Among other things, it features a redesigned editor and website, and allows you to edit projects directly from your web browser.

This module (*tutorial*) is one in a collection of modules designed to help beginners of all ages (*8 and up*) learn how to create the code for computer programs using [Scratch 2.0](#). Information is provided not only for the beginners themselves but also for their parents and teachers where appropriate.

The purpose of this module is to teach you how to use the **forever** block and the **repeat until** block to create loop structures in Scratch.

In this module, I will present and explain three Scratch programs. One program uses a **forever** block along with an **if** block to cause a sprite to move in a small circle while the space bar is pressed. You will learn that it is not possible to connect another block to the bottom of a **forever** block. Therefore, if you use a **forever** block, it must be the last block in the script in which it is contained.

The second program uses a **forever** block along with an **if** block to create a counter loop. That program will be used as the jumping-off point for a discussion of the similarities and differences between a **forever** block and a **repeat** block.

The third program uses a **repeat until** block to cause a sprite to run in a small circle until the user presses the space bar. Unlike the **forever** block, it is possible to connect another block to the bottom of a **repeat until** block. Therefore, it is not necessary for a **repeat until** block to be the last block in a script.

Finally, I will provide the specifications for a student-programming project for you to demonstrate your understanding of what you learned from the three programs listed above and from earlier modules.

Viewing tip

I recommend that you open another copy of this module in a separate browser window and use the following links to easily find and view the images while you are reading about them.

Images

- [Image A](#). Pseudocode for the program named ForeverLoop01.

- [Image B](#). Program code for the Scratch v1.4 version of the program named ForeverLoop01.
- [Image C](#). Program code for the Scratch v2.0 version of the program named ForeverLoop01.
- [Image D](#). Comparison between repeat and forever/if blocks.
- [Image E](#). Program code for the program named ForeverLoop02.
- [Image F](#). Pseudocode for the program named ForeverLoop02.
- [Image G](#). Pseudocode for the program named RepeatUntil01.
- [Image H](#). Program code for the program named RepeatUntil01.
- [Image I](#). Final screen output from the program named RepeatUntil01.
- [Image J](#). Blocks that can be used to create loop structures in v2.0.

General background information

In an earlier module, I told you that any programming logic problem could be solved using an appropriate combination of only three programming structures, none of which are complicated. The three structures are known generally as:

- The sequence structure.
- The selection or decision structure.
- The loop, repetition, or iteration structure.

I have explained the sequence and selection structures and some of the loop structures in earlier modules. I will continue to concentrate on the loop structure in this module.

Preview

In this module, I will present and explain the following three Scratch programs:

- **ForeverLoop01** - This program uses a **forever** block in conjunction with an **if block** to cause a sprite to move in a small circle while the space bar is pressed.
- **ForeverLoop02** - This program illustrates the creation of a counter loop using a **forever** block in conjunction with an **if** block.

- **RepeatUntil01** - This program uses a **repeat until** block to cause a sprite to run in a small circle until the user presses the space bar.

In addition, I will provide the specifications for a student-programming project for you to demonstrate your understanding of what you learned from the three programs listed above and from earlier modules.

Discussion and sample code

The program named ForeverLoop01

This program illustrates the use of a **forever** block in conjunction with an **if** block. When the user clicks the green flag, the program implements the pseudocode shown in [Image A](#).

Image A. Pseudocode for the program named ForeverLoop01.

Figure

Move the cat to the center of the stage.
Turn the cat to face to the right.

```
while(true){//infinite loop
  if(space bar is pressed){
    Change cat to next costume image
    Move cat forward 10 steps.
    Wait 0.1 seconds.
    Turn cat clockwise 15 degrees.
  }//end if statement
}//end infinite while loop
```

Image A. Pseudocode for the program named
ForeverLoop01.

If the space bar is pressed, the cat moves around the circumference of a small circle, appearing to walk due to the visual effect produced by

switching between two images (*known in Scratch as costumes*) . As explained in a previous module, the two costumes were designed to produce the illusion of walking.

Program code for the Scratch v1.4 version of the program named ForeverLoop01

The code for the older Scratch v1.4 version of this program is shown in [Image B](#).

Image B. Program code for the Scratch v1.4 version of the program named ForeverLoop01.

Figure



Image B. Program code for the Scratch v1.4 version of the program named ForeverLoop01.

I am showing you the code in [Image B](#) for one purpose and one purpose only. The orange block with the label "*forever if*" no longer exists in Scratch v2.0.

Older programs that were written in v1.4 and published on the Scratch website were automatically converted to the Scratch v2.0 version, as shown in [Image C](#) when the website made the transition from v1.4 to v2.0.

Program code for the Scratch v2.0 version of the program named ForeverLoop01

The code for the Scratch v2.0 version of this program is shown in [Image C](#). Note that this version replaces the **forever if** block with a **forever** block and embeds an **if** block in the **forever** block.

Except for the **forever** and **next costume** blocks, none of the blocks in [Image C](#) are new to this module.

Image C. Program code for the Scratch v2.0 version of the program named ForeverLoop01.

Figure

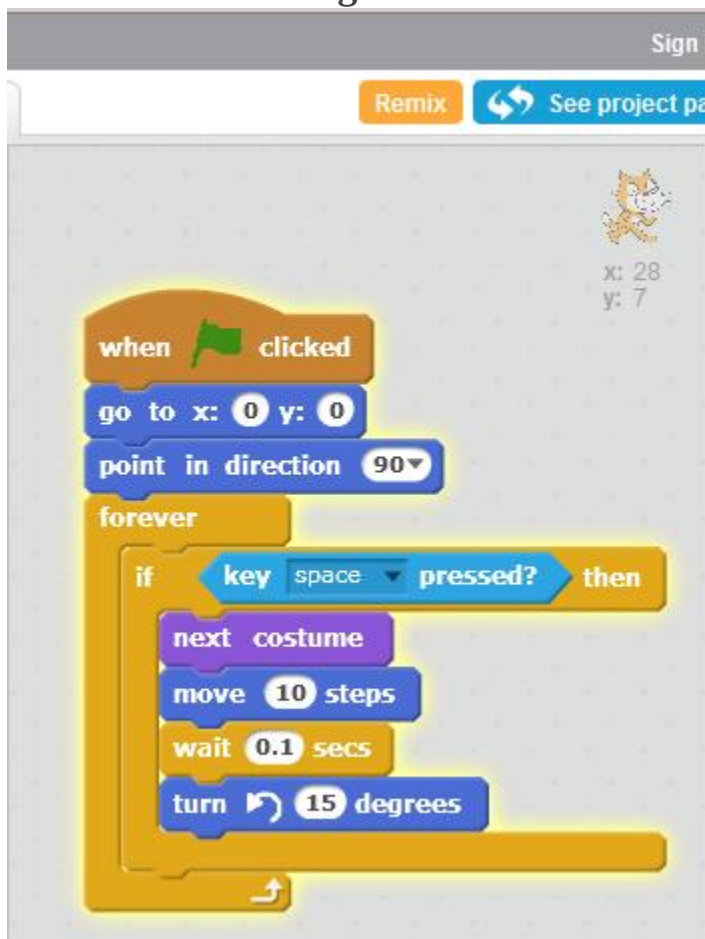


Image C. Program code for the Scratch v2.0 version of the program named ForeverLoop01.

The last block in the script

The bottom of the **forever** block is smooth. Nothing can be connected to the bottom of the block. Therefore, it must be the last block in the script that contains it.

The *next costume* block

An earlier module used a **switch to costume** block to cause the existing image of a sprite to be replaced by a different image. That block requires the programmer to specify the new costume by name each time a change in costume is needed.

The **next costume** block used in [Image C](#) is easier to use from a programming standpoint. As explained in the previous module, there is a list of one or more costumes associated with each sprite. The **next costume** block causes the sprite to switch from the current costume to the next costume in the list. If the current costume is the last costume in the list, the block causes the sprite to switch back up to the first costume in the list.

The *forever* block

Code inside a **forever** block will run forever unless

- some code inside the block causes it to stop running,
- the user clicks the red stop button on the stage, or
- the user shuts down the web page.

This is commonly referred to as an *infinite loop* .

An infinite loop is not always a bad thing. For example, many games run inside an infinite loop and that is also the case with this program. However, even though the program is running (*after you click the green flag*) , you don't see anything happening unless you press the space bar. Pressing the space bar causes the cat to run around in a small circle.

Be careful and don't leave this program running in your browser unless that is your plan. You may find that while it is running, everything else on your computer runs much more slowly than normal.

An if block

As you can see in [Image C](#) , an **if** block is embedded inside the **forever** block. During each iteration of the loop, a test is made to determine if the space key is pressed. Each time it performs the test and finds the condition to be true, the code inside the **if** block is executed. If the condition is false, the code inside the **if** block is not executed.

When the condition is true

The condition is true if the space bar is pressed when the test is made. When the condition is found to be true, the sprite:

- Changes to the next costume in the list. (*In this program the list contains two costumes.*)
- Moves ten steps forward relative to its own orientation.
- Pauses for 0.1 seconds.
- Turns fifteen degrees counter-clockwise around its own center of rotation.

No screen shot of the output

Because this program is animated, a screen shot of the Stage wouldn't impart much information so I won't show you one. Basically, it would look very similar to [Image I](#) without the thought bubble. You can copy the code from [Image C](#) and insert it into your own program. Also, the program has been posted online so that you can run it from there. (See the link to the online version in [Resources](#).)

The program named ForeverLoop02

This program illustrates another usage of a **forever** block with an embedded **if** block. The behavior of this program is very similar to a program that I explained in an earlier module using the **repeat** block. However, there are major differences between the **forever/if** combination of blocks and the **repeat** block. Some of those differences are illustrated in [Image D](#), which shows a **repeat** block at the top and a **forever** block with an embedded **if** block near the center.

Image D. Comparison between repeat and forever/if blocks.

Figure

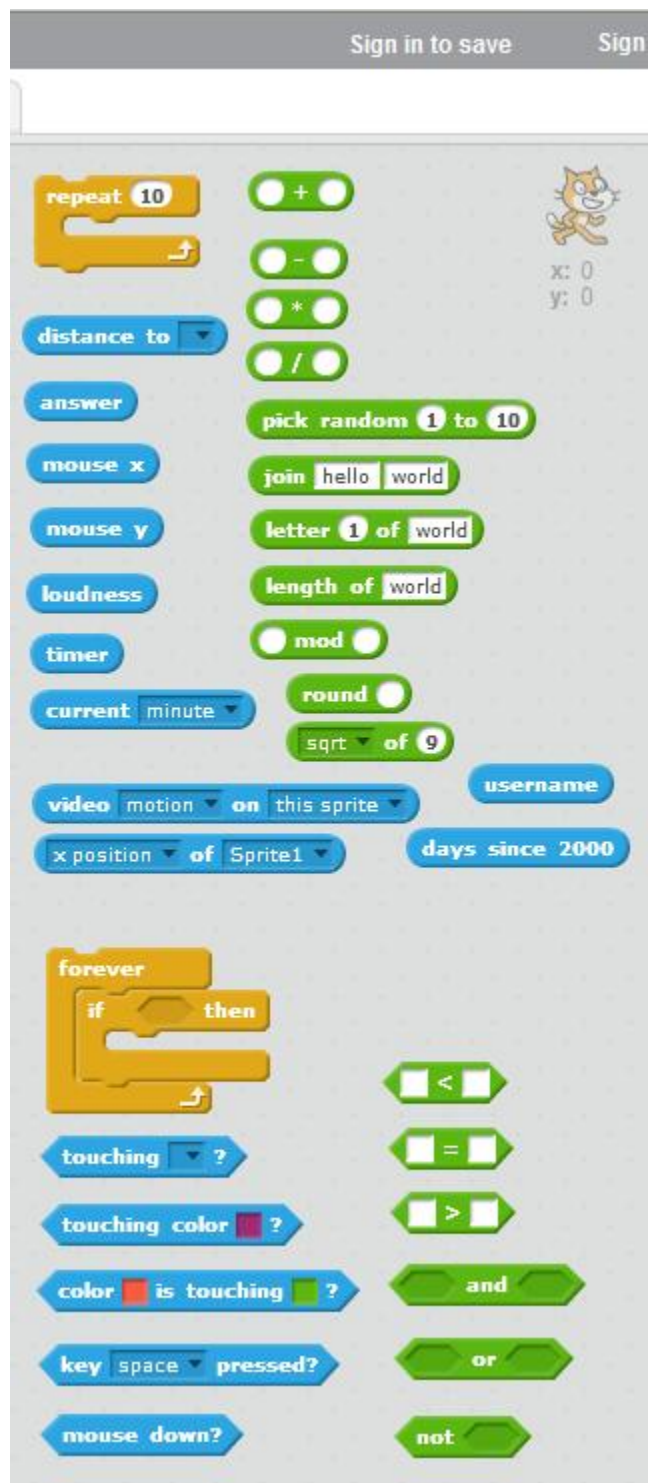


Image D. Comparison between repeat and forever/if blocks.

Comparison between *repeat* and *forever/if* blocks

One of the major differences is the shape of the blocks that can be dropped into the pockets on the **repeat** block and the **if** block.

The blocks with the rounded ends in the upper half of [Image D](#), which generally produce numeric results, cannot be dropped into the pocket with the pointed ends on the **if** block. The **if** block requires a boolean value of true or false.

However, the blocks with the rounded ends can be dropped into the pocket with rounded ends on the **repeat** block, which requires a numeric value.

The blocks with the pointed ends in the lower half of [Image D](#) can be dropped into the pocket on the **if** block. Generally speaking, those blocks produce a boolean result of true or false.

Curiously, the blocks with the pointed ends can also be dropped into the pocket with the rounded ends on the **repeat** block. That pocket needs a numeric value to specify how many times its embedded code will be executed. My unconfirmed guess is that when one of the boolean blocks with the pointed ends is dropped into a **repeat** block, a false value is treated as 0 and a true value is treated as 1.

No drop-in required for a *repeat* block

The **repeat** block doesn't require that any other block be dropped into the pocket to make it fully functional. All that is required is that a literal value be entered into the pocket, or that the default literal value be accepted. Used in this way, the **repeat** block will cause the code inside the block to be executed a number of times matching the literal value.

A drop-in is required for the *if* block

On the other hand, the **if** block requires that another block, which evaluates to either true or false, be dropped into the pocket to make it functional.

The *forever* block is more versatile than the *repeat* block

In this program, I used a counter variable in conjunction with a **less than** block to produce a boolean result, thus causing the behavior of the **forever/if** block combination to be similar to the behavior of a **repeat** block. However, the **forever** block is much more versatile. The **forever/if** combination can be used to execute a series of code blocks on the basis of any block that you can construct that will evaluate to true or false.

Can't connect to the bottom

There is one important stipulation, which I mentioned earlier. The bottom of the **forever** block is smooth and it is not possible to connect another block to the bottom of the **forever** block. Therefore, it must always be the last block in a script. Also, a script that ends with a **forever** block will continue to execute *forever* unless the program is manually terminated. That may or may not be a good thing.

Program code for the program named ForeverLoop02

The code for this program is shown in [Image E](#).
Image E. Program code for the program named ForeverLoop02.

Figure

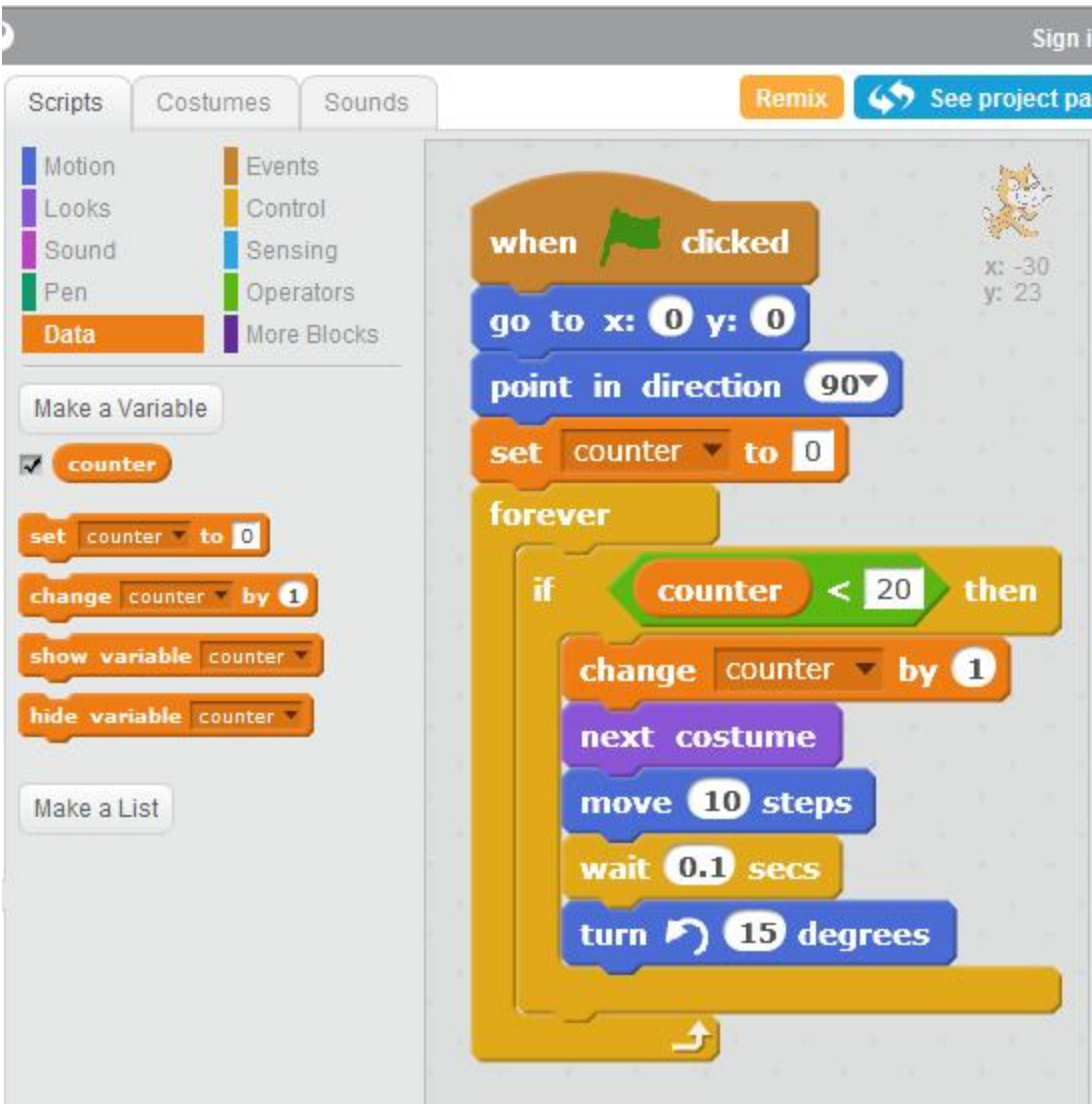


Image E. Program code for the program named ForeverLoop02.

Pseudocode for the program named ForeverLoop02

When the user clicks the green flag, the program implements the pseudocode shown in [Image F](#).

Image F. Pseudocode for the program named ForeverLoop02.

Figure

Move the cat to the center of the stage.
Turn the cat to face to the right.
Set counter to 0.

```
while(true){//infinite loop
  if(counter < 20){
    Increment counter by 1.
    Change to next costume image
    Move cat forward 10 steps.
    Wait 0.1 seconds.
    Turn cat clockwise 15 degrees.
  }//end if statement
}//end infinite while loop
```

Image F. Pseudocode for the program named
ForeverLoop02.

A counter variable

The program creates a variable named **counter** and initializes its value to zero. The code in the **if** block increments the counter by one during each iteration. For a counter limit value of 20, the code inside the **if** block is executed 20 times.

When the user clicks the green flag, the first 20 iterations of the **forever** loop cause the cat to walk about 80-percent of the way around the circumference of a small circle.

*Note that the **forever** loop continues to iterate even after the conditional clause in the **if** block goes to false. Be sure to click the red stop button in the upper-right corner of the stage to*

*terminate the **forever** loop. Otherwise, you may find other applications on your computer running very slowly.*

No screen shot of the output

Again, because this program is animated, a screen shot of the Stage wouldn't impart much information so I won't show you one. You can copy the code from [Image E](#) and run the program yourself. Also, the program has been posted online so that you can run it from there. (See the link to the online version in [Resources](#).)

The program named RepeatUntil01

This program uses a **repeat until** block to construct a loop structure. This block makes it possible to create a loop that is similar, but different from a **do-while** loop in Java.

Program behavior

When the user clicks the green flag, the program implements the pseudocode shown in [Image G](#).

Image G. Pseudocode for the program named RepeatUntil01.

Figure

Move the cat to the center of the stage.
Turn the cat to face to the right.

```
repeat until user presses space bar{  
  Change to next costume image  
  Move cat forward 10 steps.  
  Wait 0.1 seconds.  
  Turn cat clockwise 15 degrees.
```

```
}//end repeat until block
```

Move the cat to the center of the stage.

Turn the cat to face to the right.

Cause the cat to think (Hmm...)

Image G. Pseudocode for the program named
RepeatUntil01.

Program code for the program named RepeatUntil01

The code for this program is shown in [Image H](#).

Image H. Program code for the program named RepeatUntil01.

Figure

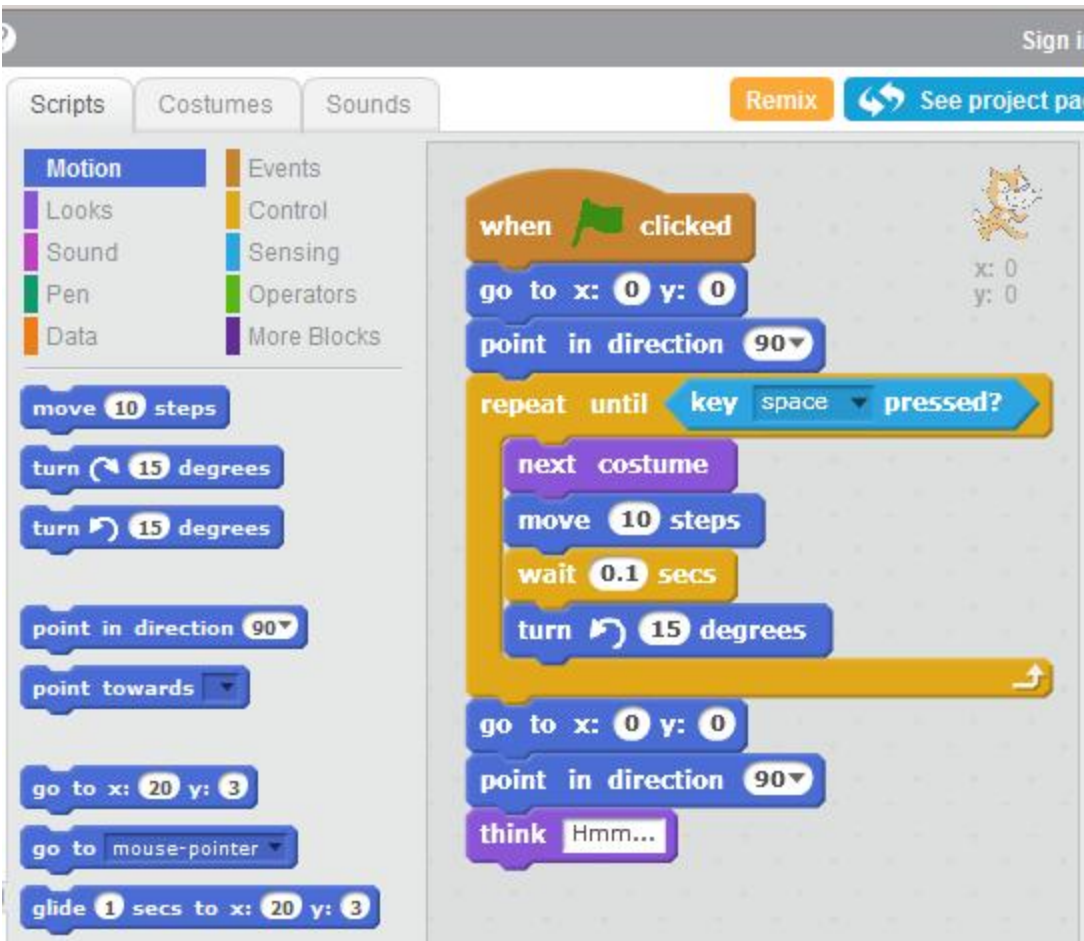


Image H. Program code for the program named RepeatUntil01.

What does the cat do?

When the user clicks the green flag, the cat

- Goes to the center of the screen.
- Turns to face to the right.
- Starts running in a circle.

The cat continues to run in the circle until the user presses the space bar, at which point the **repeat until** loop terminates.

On my computer, you must hold the space bar down for a short period of time. Simply tapping the space bar doesn't work.

Then the cat moves back to the center facing to the right where it executes a **think** block, producing the output shown in [Image I](#).

Image I. Final screen output from the program named RepeatUntil01.

Figure



Image I. Final screen output from the program named RepeatUntil01.

Doesn't have to be the last block in the script

Unlike the **forever** block, it is possible to connect another block to the bottom of a **repeat until** block. Therefore, it is not necessary for a **repeat until** block to be the last block in a script. This means that a program can continue to execute a **repeat until** block until a specified condition becomes true, and then move on to execute additional code.

This structure is more similar to the structures typically found in modern programming languages than are the structures required by the **forever** block. However, most modern programming structures repeat something **while** a condition is true instead of repeating it **until** a condition becomes true.

Available online

At the real risk of becoming really boring, I will tell you again that you can copy the code from [Image H](#) and run your own copy of the program. Also, the program has been posted online so that you can run it from there. (See the link to the online version in [Resources](#).)

The loop blocks

The three blocks shown in [Image J](#) are the only blocks available to create loop structures in Scratch version 2.0.

*(The **forever if** block that was available in v1.4 is no longer available in v2.0. However, its functionality can be approximated by embedding an **if** block in a **forever** block.)*

Image J. Blocks that can be used to create loop structures in v2.0.

Figure



Image J. Blocks that can be used to create loop structures in v2.0.

In this and the previous program, I have provided discussions and sample programs using the three blocks in [Image J](#) as well as a discussion and sample program for a **forever** block with an embedded **if** block.

Run the programs

I encourage you to use the information provided above to write and run these three programs. Experiment with the code, making changes, and observing the results of your changes. Make certain that you can explain why your changes behave as they do.

Just for fun, see if you can cause the cat to change color as it marches around in the circle.

I also encourage you to write the program described below.

Student programming project

Write a program named **ForeverLoop03** that is very similar to the program named **ForeverLoop01** . Once again, the cat marches in a circle when the user holds down the space bar (*after first clicking the green flag*) . However, this program also produces an audio output that is a nice drumbeat and the cat marches in synchronism with the drumbeat.

My version of this program has been posted online so that you can run it from there. (See [Resources](#) for the link to the online version.)

Because of the speed of my computer and the speed of my Internet connection, the synchronization of the drumbeat and the animation in v2.0 is not as good as it was when running the program locally under v1.4. Losing the ability to run Scratch programs locally is one disadvantage of the changes that were made in v2.0.

Summary

In this module, I presented and explained three Scratch programs. One program uses a **forever** block in conjunction with an embedded **if** block to cause a sprite to move in a small circle while the space bar is pressed. I explained that it is not possible to connect another block to the bottom of a **forever** block. Therefore, if you use a **forever** block, it must be the last block in the script in which it is contained.

The second program uses a **forever** block in conjunction with an embedded **if** block and a counter variable to create a counter loop. That program was

used as the jumping-off point for a discussion of the similarities and differences between a **forever/if** block combination and a **repeat** block.

The third program uses a **repeat until** block to cause a sprite to run in a small circle until the user presses the space bar. I pointed out that unlike the **forever** block, it is possible to connect another block to the bottom of a **repeat until** block. Therefore, it is not necessary for a **repeat until** block to be the last block in a script.

Finally, I provided the specifications for a student-programming project for you to demonstrate your understanding of what you learned from the three programs listed above and from earlier modules.

What's next?

One of the new features of Scratch v2.0 is the ability for you to create your own blocks. The next module will discuss that feature.

Resources

General resources

- [Scratch home](#)
- [Scratch tutorials](#)
- [Scratch explanatory video](#)
- [Scratch tutorials from MIT](#)
- [Scratch tutorial - Digital Art for All](#)
- [Scratch for Budding Computer Scientists](#)
- [Learn Scratch at learnscratch.org](#)
- [Scratch Tutorial - Space Shuttle Mission STS-2020](#)
- [SCRATCH TUTORIALS ICT In Primary Education 2012-2013](#)
- [Scratch Wiki](#)
- [Scratch WikiTable of ContentsWebsite - Scratch Wiki](#)
- [Scratch WikiTable of ContentsTutorials - Scratch Wiki](#)
- [Scratch WikiTable of ContentsProgram - Scratch Wiki](#)
- [Tutorial about Variables - Scratch Wiki](#)

- [Scratch support MIT web site](#)
- [Scratch resources at Scratch.ie](#)
- [ScratchEd Resources](#)
- [Scratch project Day Dream](#)
- [Scratch project Son of String Art](#)
- [Scratch Project Scratch Tutorial](#)
- [DeMorgan's theorem](#)

Programs used in this series

- [Variable01](#) - Online version of program
- [Variable02](#) - Online version of student-programming project
- [Variable03](#) - Online version of student-programming project
- [IfSimple01](#) - Online version of program
- [IfWithVar01](#) - Online version of student-programming project
- [Arithmetic01](#) - Online version of program
- [Arithmetic02](#) - Online version of student-programming project
- [Relational01](#) - Online version of program
- [Relational02](#) - Online version of student-programming project
- [Logical01](#) - Online version of program
- [Logical02](#) - Online version of student-programming project
- [Logical03](#) - Online version of student-programming project
- [ForLoop01](#) - Online version of program
- [ForLoop02](#) - Online version of program
- [ForLoop03](#) - Online version of student-programming project
- [ForeverLoop01](#) - Online version of program
- [ForeverLoop02](#) - Online version of program
- [RepeatUntil01](#) - Online version of program
- [ForeverLoop03](#) - Online version of student-programming project

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Scr0380: The forever and repeat until loops in Scratch 2.0
- File: Scr0380.htm
- Published: 05/19/13

Note: Disclaimers:

Financial : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.

I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module. In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.

Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Scr0390: Custom blocks in Scratch 2.0

The purpose of this module is to explain the use of functions, methods, and procedures in other programming languages, and to show you how to accomplish much of that using the new custom block feature of Scratch 2.0.

Table of Contents

- [Preface](#)
 - [Viewing tip](#)
 - [Images](#)
- [Introduction](#)
 - [Calculate the square root](#)
 - [Oops, need to do it all over again](#)
 - [Is there a better way?](#)
 - [A programming module provides a better way.](#)
 - [Library functions](#)
 - [Passing parameters](#)
 - [Make the function general](#)
 - [Pass me the number please](#)
 - [Passing parameters](#)
 - [Returning values](#)
 - [Performing an action](#)
 - [Sending back an answer](#)
- [Creating your own custom blocks](#)
- [Sample programs](#)
 - [CustomBlockTest01](#)

- [A global variable](#)
- [The custom block](#)
- [The driver script](#)
- [The program output](#)
- [CustomBlockTest02](#)
 - [The custom block](#)
 - [The driver script](#)
 - [The program output](#)
- [Run the programs](#)
- [Student programming project](#)
- [Resources](#)
 - [General resources](#)
 - [Programs used in this series](#)
- [Miscellaneous](#)

Preface

[Scratch 2.0](#) (*released May 9, 2013*) is the second major version of Scratch to be released during the life of the product. Among other things, it features a redesigned editor and website, and allows you to edit projects directly from your web browser.

This module (*tutorial*) is one in a collection of modules designed to help beginners of all ages (*8 and up*) learn how to create the code for computer programs using [Scratch 2.0](#). Information is provided not only for the beginners themselves but also for their parents and teachers where appropriate.

The purpose of this module is to explain the use of functions, methods, and procedures in other programming languages, and to show you how to accomplish much of that using the new custom block feature of Scratch 2.0.

Viewing tip

I recommend that you open another copy of this module in a separate browser window and use the following links to easily find and view the images while you are reading about them.

Images

- [Image A](#). The More Blocks toolbox.
- [Image B](#). Access to Scratch 2.0 math functions.
- [Image C](#). Mathematical functions available in Scratch 2.0.
- [Image D](#). Program code for CustomBlockTest01.
- [Image E](#). Output from CustomBlockTest01.
- [Image F](#). Program code for CustomBlockTest02.

Introduction

Modular programming code (*methods, subroutines, functions, procedures, or whatever you choose to call them*) have been used in computer programming since the early days of programming. Such coding modules are often called by these and various other names.

The new ability in Scratch 2.0 to create custom blocks can be used to provide much of the same modular functionality.

Calculate the square root

Suppose that your program needs to calculate the square root of a number.

In case you don't know what it means to compute the square root of a number, don't worry about it. Suffice it to say that it is a fairly complex mathematical process that requires a non-trivial amount of computer program code to accomplish.

Referring back to your high-school algebra book, you could refresh your memory on how to calculate a square root. Then you could construct the algorithm describing that process. Having the algorithm available, you could write the code to calculate the square root and insert that code into your program code. Then you run your program.

If you did it all correctly, your program should calculate the square root. *(For reasons that will become apparent later, I will refer to the code that you inserted as in-line code.)*

Oops, need to do it all over again

Suppose that further on in your program you discover that you need to calculate the square root of another number. And later, you discover that you need to calculate the square root of still another number. Obviously, with a few changes, you could copy your original code and insert it as *in-line* code at each location in your program where you need to calculate the square root of a number.

Is there a better way?

However, after doing this a few times, you might start asking if there is a better way. The answer is "yes, there is a better way."

A programming module provides a better way

The better way is to create a separate program module that has the ability to calculate the square root and to make that module available for use as a helper to your main program each time your main program needs to calculate a square root. In some programming languages, this separate program module is often called a *function* or a *method*. In scratch 2.0, it doesn't have a specific name. I will refer to it as a *custom block*. You create a custom block in Scratch 2.0 by selecting the **More Blocks** toolbox shown in [Image A](#).

Image A. The More Blocks toolbox.

Figure

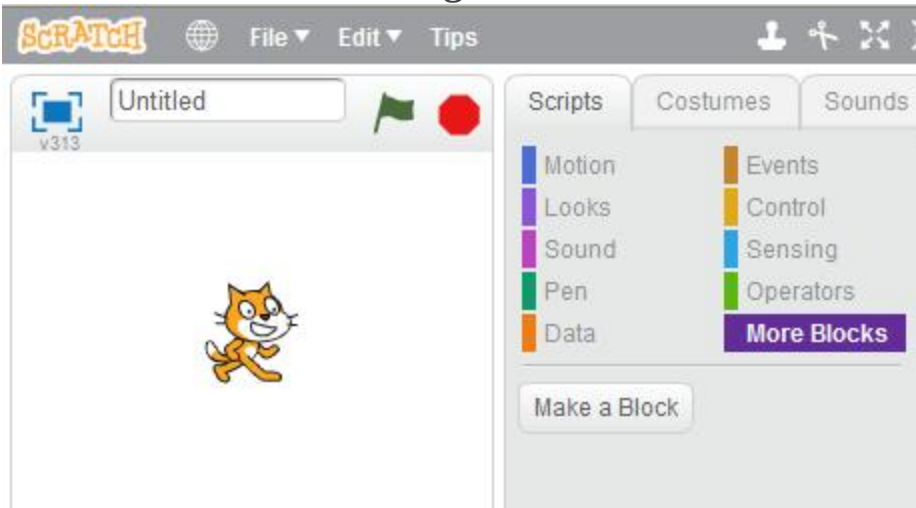


Image A. The More Blocks toolbox.

Library functions

Most modern programming languages provide a large number of pre-written functions that are already available for your use.

In addition to the library functions that are already available, if you need a function to perform some operation and there is no library function already available to perform that operation, you can write your own function.

A similar library for mathematical functions exists in Scratch 2.0. The library is accessed using the bottom block in the **Operators** toolbox with the selection `sqrt` shown in [Image B](#).

Image B. Access to Scratch 2.0 math functions.

Figure



Image B. Access to
Scratch 2.0 math
functions.

[Image C](#) shows the different mathematical function selections that are available in the pull down list of the block. (Note that the fourth one from the top is the **sqrt** function.)

Image C. Mathematical functions available in Scratch 2.0.

Figure



Image C.
Mathematical
functions available in
Scratch 2.0.

Passing parameters

Make the function general

Normally, in the case of designing and writing a function such as one that can calculate the square root of a number, it is desirable to write it in such a

way that it can calculate the square root of any number (*as opposed to only one specific number*) . This is accomplished through the use of something called parameters.

Pass me the number please

The process of causing a function to be executed is commonly referred to as *calling* the function.

When your program calls the square-root function, it will need to tell the function the value of the number for which the square root is needed. In general, many functions will require that you provide certain kinds of information when you call them. The code in the function needs this information to be able to accomplish its purpose.

Passing parameters

This process of providing information to a function when you call it is commonly referred to as *passing parameters* to the function. For the square-root function, you need to pass a parameter whose value is the value of the number for which you need the square root.

*You call the built-in square root function in Scratch 2.0 by dragging the bottom block shown in [Image B](#) into the programming panel and selecting **sqrt** in the pull down list. You pass the parameter by entering a literal value or dragging a variable block into the white area that contains the 9 character in [Image B](#).*

Returning values

A function or method will usually

- Perform an action,
- Send back an answer
- Some combination of the two

Performing an action

You might think of *performing an action* as something like causing a Scratch sprite to move to the right and change color.

None of the Scratch 2.0 mathematical functions shown in [Image C](#) perform an action. One of the purposes of this module is to teach you how to create custom blocks that do perform an action.

Sending back an answer

On the other hand, a function that is designed to calculate the square root of a number needs to be able to send the square-root value back to the program that called the function. After all, it wouldn't be very useful if the function calculated the square root and then kept it a secret. The process of sending back an answer is commonly referred to as *returning a value* .

All of the Scratch mathematical functions shown in [Image C](#) return a value.

It appears that there is no straightforward way for you to cause a custom block that you design to return a value. However, I will show you a workaround for that situation later.

Creating your own custom blocks

The best explanation that I have found for the mechanics of creating a custom block is contained in the document titled [How Do I Use the New Blocks?](#) on the MIT website. Go to that webpage and scroll down to the section titled *Make Your Own Blocks* . Follow the instructions that you find there to create your own custom blocks.

Sample programs

With that as an introduction, this section will present and explain two Scratch 2.0 programs built around custom blocks.

CustomBlockTest01

This is the simplest custom block that I could come up with that:

- Receives an incoming parameter
- Uses the value of that parameter to compute and return a value

This program contains the two scripts and a variable shown in [Image D](#). These scripts are tied to the cat sprite.

Image D. Program code for CustomBlockTest01.

Figure

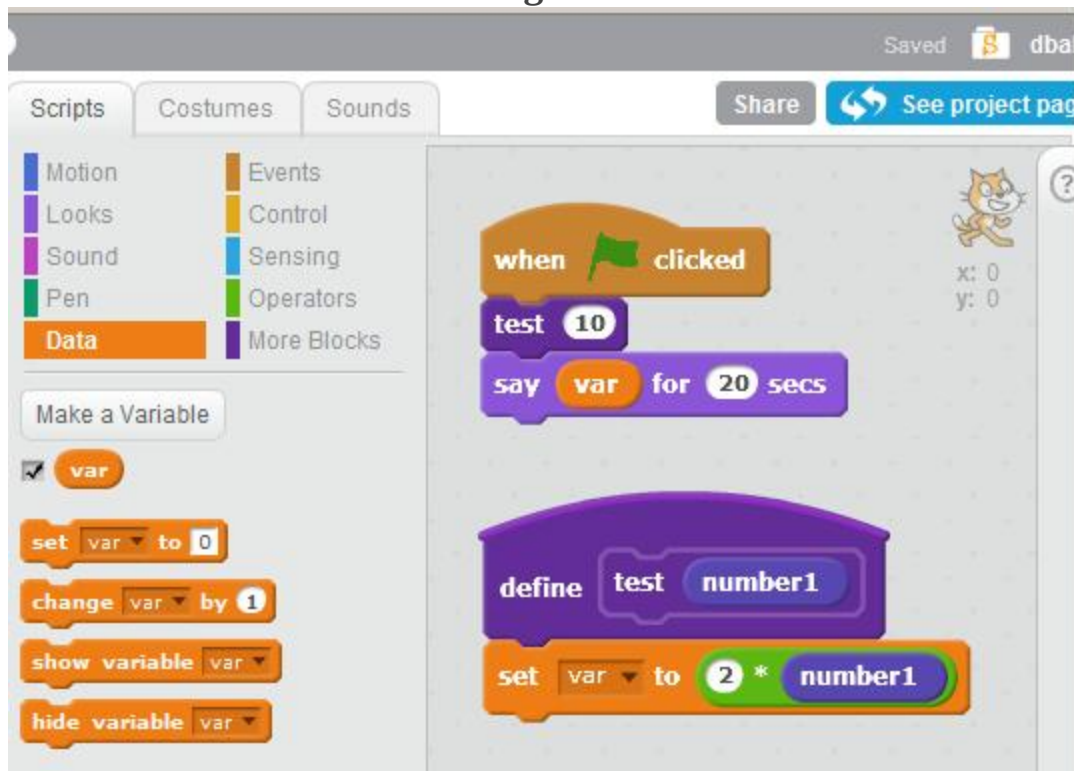


Image D. Program code for CustomBlockTest01.

A global variable

The variable named **var** shown on the left in [Image D](#) is a global variable. That means that it is accessible by any code in the program.

The custom block

The custom block named **test** is shown in the bottom script in [Image D](#). When called, this block receives an incoming numeric parameter known locally as **number1**. The code in the block multiplies that value by a factor of 2 and stores the resulting product in the variable named **var**, making it available to the code that called the custom block.

The driver script

The code that calls the custom block is shown in the top script in [Image D](#). (*I will refer to this as the driver script.*) When the green flag is clicked, the custom block named **test** is called by the driver, passing a numeric value of 10 as a parameter. When the **test** block terminates, control is returned to the driver. The driver script causes the cat sprite to display the value stored in the variable named **var** for 20 seconds. Then the program terminates.

The program output

The program output showing the cat on the stage is shown in [Image E](#). Image E. Output from CustomBlockTest01.

Figure



Image E. Output from CustomBlockTest01.

A numeric value of 10 is passed to the **test** block by the driver. Code in the **test** block multiplies that value by a factor of 2 storing the product value of 20 in the variable named **var** . Code in the driver then causes that value to be displayed by the cat sprite on the stage. Thus, the program displays the value 20 as shown in [Image E](#) .

CustomBlockTest02

This program illustrates some of the benefits of custom blocks. The two scripts that comprise the program are shown in [Image F](#) .

Image F. Program code for CustomBlockTest02.
Figure

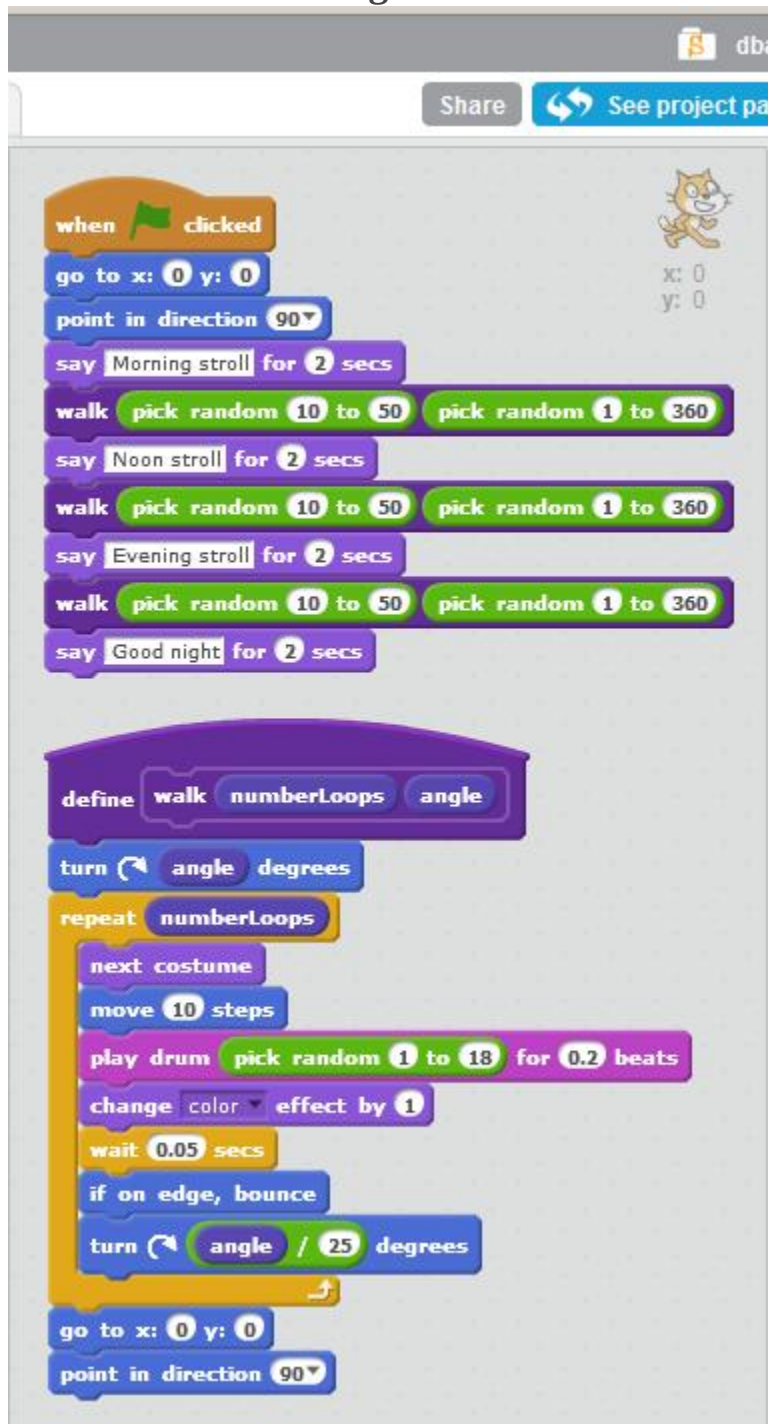


Image F. Program code for
CustomBlockTest02.

The top script in [Image F](#) is the driver script. This code calls the custom block (*shown in the bottom script*) three times.

The custom block

The purpose of the custom block is to cause the cat to take a walk with the duration of the walk and the initial direction of the walk determined by a pair of incoming parameters.

As you will see later, the cat is facing to the right each time the custom block named **walk** is called. The incoming parameter named **angle** is used to cause the cat to turn and face in a particular direction before beginning the walk. The same parameter is used later in the walk to cause the cat to make a much smaller turn in the same direction at the end of each segment of the walk. Thus, the cat tends to walk in a somewhat circular path unless it collides with the edge of the stage, in which case it bounces off the edge.

The incoming parameter named **numberLoops** is used to determine the number of segments in the walk or the duration of the walk.

The last two blocks in the custom block cause the cat to return to the center of the stage and face to the right.

By now you shouldn't need an explanation for the remaining code in the custom block.

The driver script

The driver script initially causes the cat to move to the center of the screen and face to the right.

Then it executes three pairs of blocks in sequence. The first block in each pair causes the cat to say "Morning stroll", "Noon stroll", or "Evening stroll".

The second block in each pair is a call to the custom block named **walk** . Each time the **walk** block is called, it is passed random values for **numberLoops** and **angle** . Therefore, the duration and direction of each walk will be different each time you run the program.

The program output

Because of the dynamic nature of the program, there is no point in me presenting a static snapshot of the output. Instead, I will simply suggest that you run the program online at [CustomBlockTest02](#) and observe the output.

Run the programs

I encourage you to use the information provided above to write and run these two programs. Experiment with the code, making changes, and observing the results of your changes. Make certain that you can explain why your changes behave as they do.

Student programming project

It's time for you to be creative. Create a project of your own, or perhaps remix an existing project from the Scratch website. Break the project into a driver script and a custom block.

Resources

General resources

- [Scratch home](#)
- [Scratch tutorials](#)
- [Scratch explanatory video](#)
- [Scratch tutorials from MIT](#)
- [Scratch tutorial - Digital Art for All](#)
- [Scratch for Budding Computer Scientists](#)
- [Learn Scratch at learnscratch.org](#)

- [Scratch Tutorial - Space Shuttle Mission STS-2020](#)
- [SCRATCH TUTORIALS ICT In Primary Education 2012-2013](#)
- [Scratch Wiki](#)
- [Scratch WikiTable of ContentsWebsite - Scratch Wiki](#)
- [Scratch WikiTable of ContentsTutorials - Scratch Wiki](#)
- [Scratch WikiTable of ContentsProgram - Scratch Wiki](#)
- [Tutorial about Variables - Scratch Wiki](#)
- [Scratch support MIT web site](#)
- [Scratch resources at Scratch.ie](#)
- [ScratchEd Resources](#)
- [Scratch project Day Dream](#)
- [Scratch project Son of String Art](#)
- [Scratch Project Scratch Tutorial](#)
- [DeMorgan's theorem](#)
- [How Do I Use the New Blocks?](#)

Programs used in this series

- [Variable01](#) - Online version of program
- [Variable02](#) - Online version of student-programming project
- [Variable03](#) - Online version of student-programming project
- [IfSimple01](#) - Online version of program
- [IfWithVar01](#) - Online version of student-programming project
- [Arithmetic01](#) - Online version of program
- [Arithmetic02](#) - Online version of student-programming project
- [Relational01](#) - Online version of program
- [Relational02](#) - Online version of student-programming project
- [Logical01](#) - Online version of program
- [Logical02](#) - Online version of student-programming project
- [Logical03](#) - Online version of student-programming project
- [ForLoop01](#) - Online version of program
- [ForLoop02](#) - Online version of program
- [ForLoop03](#) - Online version of student-programming project
- [ForeverLoop01](#) - Online version of program
- [ForeverLoop02](#) - Online version of program
- [RepeatUntil01](#) - Online version of program

- [ForeverLoop03](#) - Online version of student-programming project
- [CustomBlockTest01](#) - Online version of program
- [CustomBlockTest02](#) - Online version of program

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Scr0390: Custom blocks in Scratch 2.0
- File: Scr0390.htm
- Published: 05/27/13

Note: Disclaimers:

Financial : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.

I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module. In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.

Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Scr0400: Highlights from Scratch 2.0

The purpose of this module is to highlight a few Scratch projects.

Table of Contents

- [Preface](#)
 - [Viewing tip](#)
 - [Images](#)
- [Discussion and sample projects](#)
 - [Fireworks by cloning](#)
 - [StarDoors01](#)
 - [Perspective03](#)
 - [Hundreds of available projects](#)
- [Resources](#)
 - [General resources](#)
 - [Programs used in this series](#)
- [Miscellaneous](#)

Preface

[Scratch 2.0](#) (*released May 9, 2013*) is the second major version of Scratch to be released during the life of the product. Among other things, it features a redesigned editor and website, and allows you to edit projects directly from your web browser.

This module (*tutorial*) is one in a collection of modules designed to help beginners of all ages (*8 and up*) learn how to create the code for computer programs using [Scratch 2.0](#). Information is provided not only for the beginners themselves but also for their parents and teachers where appropriate.

The purpose of this module is to highlight a few Scratch projects.

Viewing tip

I recommend that you open another copy of this module in a separate browser window and use the following links to easily find and view the images while you are reading about them.

Images

- [Image A](#). Screen shot from FireWorks .
- [Image B](#). Screen shot from StarDoors01.
- [Image C](#). Screen shot from Perspective03.

Discussion and sample projects

Fireworks by cloning

One of the very interesting new features of Scratch 2.0 is cloning. You will find some instructions for using this feature at [Clone Blocks](#). According to the material on that web page, the use of clone blocks:

Creates a clone of the specified sprite. (The clone is a duplicate that only lasts while the project is running.)

[Image A](#) shows a screen shot of a project named [FireWorks](#) by [Illusionist](#).
Image A. Screen shot from FireWorks.

Figure

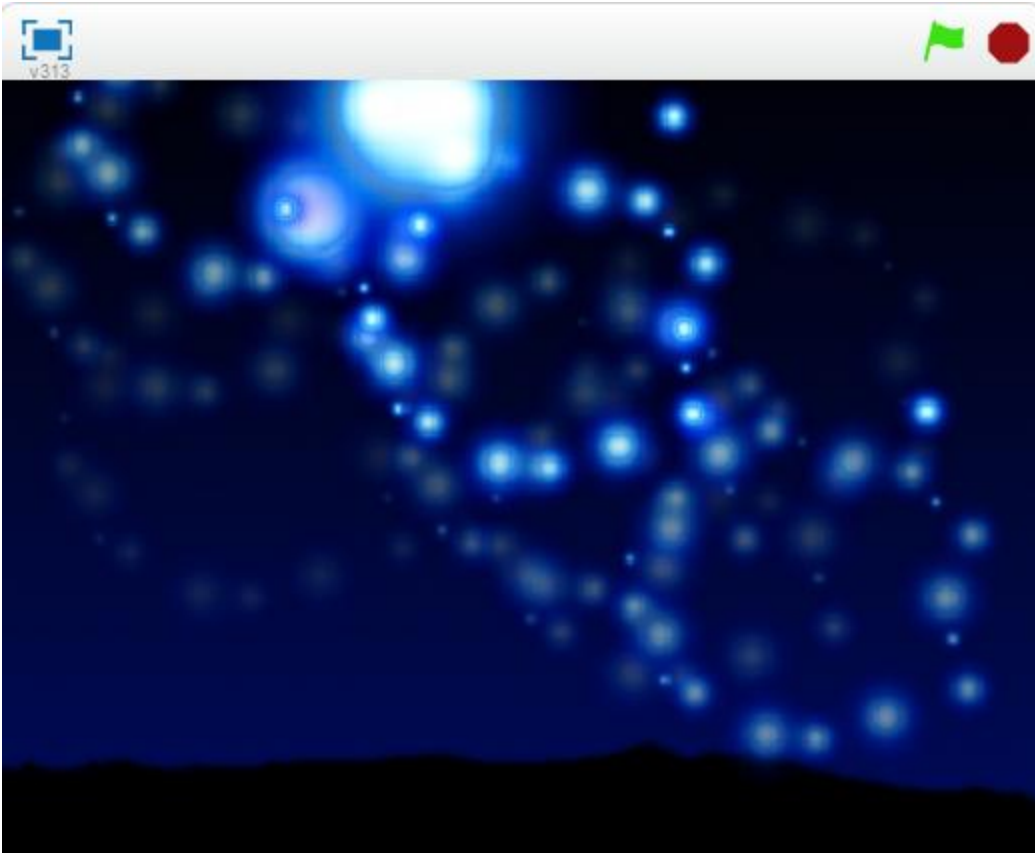


Image A. Screen shot from FireWorks.

This screen shot doesn't begin to do justice to this project. Click [here](#) to run the project online. Press the spacebar to fire a single rocket. Hold the spacebar down for a grand finale.

You may notice that I remixed this project. However, I made no changes to the original project by [Illusionist](#). My purpose in remixing the project under my user name was simply to ensure that the project will continue to be available in its current form. In the past, some of the Scratch projects that I have referred to have later been removed from the Scratch website resulting in broken links in my documents.

StarDoors01

Click [here](#) to run the project named [StarDoors01](#) by [dbal](#). A screen shot from the project is shown in [Image B](#).

Image B. Screen shot from StarDoors01.

Figure

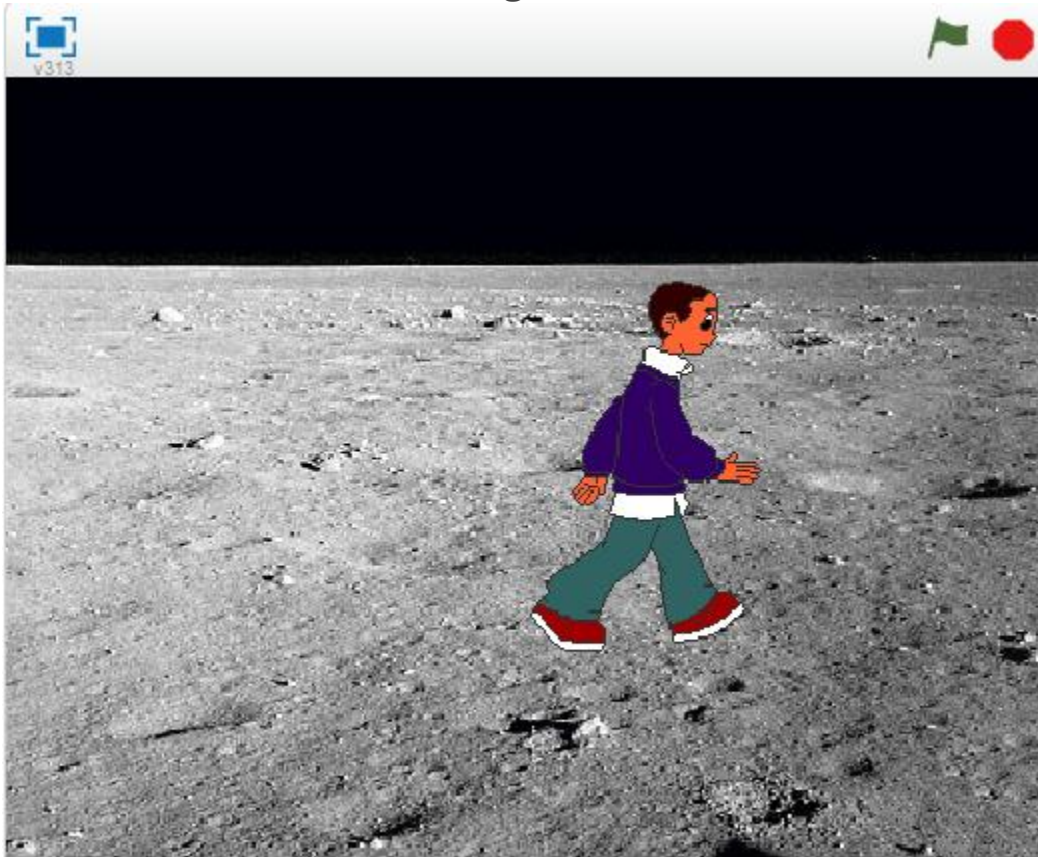


Image B. Screen shot from StarDoors01.

While the screen shot in [Image B](#) isn't all that impressive, the project illustrates several important Scratch programming concepts including frame animation, scaling, communication among sprites, etc.

Perspective03

Click [here](#) to run the project named [Perspective03](#) by [dbal](#). A screen shot of the project is shown in [Image C](#).

Image C. Screen shot from Perspective03.

Figure

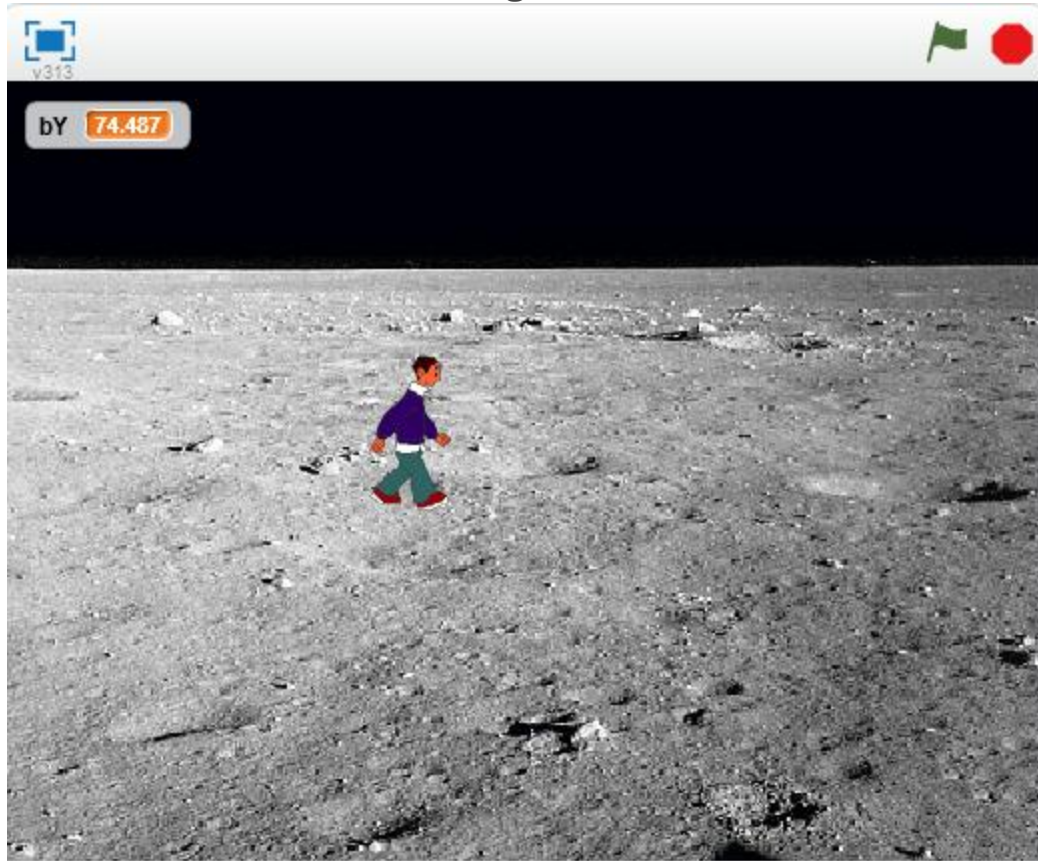


Image C. Screen shot from Perspective03.

This project continues the theme of frame animation. In this case, however, there is also the illusion of perspective. The animated sprite follows the mouse and appears to move further away from the viewer as it approaches the horizon.

Hundreds of available projects

There are hundreds, perhaps thousands of projects available on the [Scratch](#) website for you to [explore](#). Many are outstanding. Some not so good. In any event, you can examine them all for free, so you have nothing to lose. You don't even have to log in to run the projects online in your browser.

Resources

General resources

- [Scratch home](#)
- [Scratch tutorials](#)
- [Scratch explanatory video](#)
- [Scratch tutorials from MIT](#)
- [Scratch tutorial - Digital Art for All](#)
- [Scratch for Budding Computer Scientists](#)
- [Learn Scratch at learnscratch.org](#)
- [Scratch Tutorial - Space Shuttle Mission STS-2020](#)
- [SCRATCH TUTORIALS ICT In Primary Education 2012-2013](#)
- [Scratch Wiki](#)
- [Scratch WikiTable of ContentsWebsite - Scratch Wiki](#)
- [Scratch WikiTable of ContentsTutorials - Scratch Wiki](#)
- [Scratch WikiTable of ContentsProgram - Scratch Wiki](#)
- [Tutorial about Variables - Scratch Wiki](#)
- [Scratch support MIT web site](#)
- [Scratch resources at Scratch.ie](#)
- [ScratchEd Resources](#)
- [Scratch project Day Dream](#)
- [Scratch project Son of String Art](#)
- [Scratch Project Scratch Tutorial](#)
- [DeMorgan's theorem](#)
- [How Do I Use the New Blocks?](#)
- [Make Your Own Blocks](#)
- [Clone Blocks](#)
- [Explore hundreds of projects](#)

Programs used in this series

- [Variable01](#) - Online version of program
- [Variable02](#) - Online version of student-programming project
- [Variable03](#) - Online version of student-programming project
- [IfSimple01](#) - Online version of program
- [IfWithVar01](#) - Online version of student-programming project
- [Arithmetic01](#) - Online version of program
- [Arithmetic02](#) - Online version of student-programming project
- [Relational01](#) - Online version of program
- [Relational02](#) - Online version of student-programming project
- [Logical01](#) - Online version of program
- [Logical02](#) - Online version of student-programming project
- [Logical03](#) - Online version of student-programming project
- [ForLoop01](#) - Online version of program
- [ForLoop02](#) - Online version of program
- [ForLoop03](#) - Online version of student-programming project
- [ForeverLoop01](#) - Online version of program
- [ForeverLoop02](#) - Online version of program
- [RepeatUntil01](#) - Online version of program
- [ForeverLoop03](#) - Online version of student-programming project
- [CustomBlockTest01](#) - Online version of program
- [CustomBlockTest02](#) - Online version of program
- [Fireworks](#) - by [Illusionist](#)
- [StarDoors01](#) - by [dbal](#)
- [Perspective03](#) - by [dbal](#)

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Scr0400: Highlights of Scratch 2.0
- File: Scr0400.htm

- Published: 05/27/13

Note: Disclaimers:

Financial : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.

I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module. In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.

Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Scr0100: Scratch Overview

The purpose of this module is to provide a high-level overview of the Scratch programming environment.

Table of Contents

- [Preface](#)
 - [General](#)
 - [A prediction](#)
 - [Where are the little leagues of computer science?](#)
 - [Some statistics](#)
 - [Many archived projects are readily available](#)
 - [Post, review, and comment](#)
 - [A world-wide phenomenon](#)
 - [Demographics](#)
 - [Not in organized classes](#)
 - [Viewing tip](#)
 - [Images](#)
- [Programming for many purposes](#)
 - [The Scratch player applet](#)
 - [Computer art by pandalecteur](#)
 - [Click the link to run the applet](#)
 - [Be sure to close the pages](#)
 - [What if the applets don't run?](#)
 - [Anime projects](#)
 - [Game projects](#)
 - [3D games](#)
 - [A pseudo-3D animation](#)

- [An animated story](#)
 - [The tip of the iceberg](#)
- [Scratch development environment](#)
 - [A sprite-oriented programming environment](#)
 - [Costumes](#)
 - [Walking-boy costumes](#)
 - [The stage](#)
 - [Controlling the behavior of a sprite](#)
 - [Drag and drop programming](#)
 - [Costumes and sounds](#)
- [The Scratch programming language](#)
 - [Music, music, music](#)
 - [The toolbox buttons and toolbox pane](#)
 - [High-level multimedia functionality](#)
 - [Program control](#)
- [Summary](#)
- [Resources](#)
- [Miscellaneous](#)

Preface

General

This module is one in a collection of modules designed to help beginners of all ages (*8 and up*) learn how to create the code for computer programs. Information is provided not only for the beginners themselves but also for their parents and teachers where appropriate.

The purpose of this module is to provide a high-level overview of the [Scratch](#) programming environment. Future modules will provide more

detailed information about the programming environment.

While this module is not intended to exclude the "older" beginners, much of the information contained in this module may be of more interest to the parents and teachers of younger students who are wondering if they should allow the students in their charge to become involved in Scratch.

A prediction

It's probably safe to say that a large percentage of the professional athletes in the major leagues of sports are people who became involved in sports in little league when they were quite young. It's probably also safe to say that a large percentage of successful professional musicians were either enrolled in serious musical instruction at an early age, or were involved as members of garage bands by their early teens.

I predict that in ten to fifteen years, computer science professionals and professionals in a variety of computer-related fields will be the people who are currently engaged in the *little leagues of computer science*.

Where are the little leagues of computer science?

One of the most important *little leagues of computer science* is centered in the [Lifelong Kindergarten group](#) at the [MIT Media Lab](#). This group has developed and is supporting a programming environment for beginners called [Scratch](#). According to MIT,

"Scratch is a programming language that makes it easy to create your own interactive stories, animations, games, music, and art -- and share your creations on the web.

Scratch is designed to help young people (ages 8 and up) develop 21st century learning skills. As they create Scratch projects,

young people learn important mathematical and computational ideas, while also gaining a deeper understanding of the process of design."

Scratch is available free of charge and is currently available for Mac OSX, Windows, and Debian / Ubuntu.

Some statistics

As of March 15, 2013, the Scratch home page indicates that 3,171,004 projects have been created (*and that number is increasing by about 230 projects per hour or 4 projects per minute*) .

According to an [article](#) published by MIT personnel

"On the morning of May 14, 2007, the (Scratch Online Community) website was officially launched. Several news outlets and social news websites featured the Scratch website on their front pages. In a matter of hours the server and the website could not handle the traffic and the website went down several times."

If I did the arithmetic correctly, this represents an average of one new project every minute being posted on the website since the inception of the website in 2007. Of course, many other projects have been created but not published on the website.

Many archived projects are readily available

Thousands of archived projects are available for online execution and/or downloading and examination at the source code level. Thus, the amount of

resource material available to budding Scratch programmers is almost limitless.

Post, review, and comment

MIT makes it possible for every *scratcher* (as the members of the Scratch community refer to themselves) to share his or her projects for online execution and/or downloading and examination by other scratchers. MIT also makes it possible for other scratchers to review and critique the projects shared by others. Registration is free and open to everyone.

MIT seems to make a significant effort to hide the true identities of the registrants and also seems to make a significant effort to ensure that the posted material is age appropriate for middle school students. (See [Community Guidelines](#).)

An organized system for peer review and critique is provided. The system is too elaborate for me to describe. If you want to know more about it, simply register (*become a scratcher*) and take a look at it for yourself.

A world-wide phenomenon

This is not just a United States phenomenon. The scratchers are located in many different countries around the world. A forum is provided for scratchers to exchange information with one another. As of this writing, that forum is available in at least fifteen different languages.

Also, many scratchers whose first language is not English participate in the English version of the forums and provide English-language descriptions of their projects when they publish them.

Demographics

An article published around 2008 stated that based on the first five months of usage data,

"...users are primarily age 8 to 17, with a peak at age 12. A good number of users are adult computer hobbyists and educators that create projects in Scratch, even though a lot of them know other professional programming languages. Some members of the community have emerged as mentors that help the beginners and provide advice."

The article also stated,

"While 70 percent of users are male, no correlation was found between gender and the number of projects... This indicates that even though the majority of users are male, the females are as engaged in creating projects as the males."

Not in organized classes

By following the discussions in the forums, I have observed that a large percentage of scratchers are not enrolled in any sort of formal programming classes. Rather, this is something that they are doing on their own. In fact, it seems that for many scratchers, creating, sharing, and discussing projects is a form of social networking much like other students might engage in using the better known social networking sites.

Viewing tip

I recommend that you open another copy of this document in a separate browser window and use the following links to easily find and view the images while you are reading about them.

Images

- [Image 1](#). Rushes with sun by pandalecteur.
- [Image 2](#). How to draw a head by dialga..
- [Image 3](#). Marble Racer SBE 4000 by jamie.
- [Image 4](#). Perspective03 by dbal.
- [Image 5](#). Day Dream by cremeglance.
- [Image 6](#). Screen shot of the Scratch development environment.
- [Image 7](#). Costumes for a boy walking.
- [Image 8](#). The toolbox buttons and toolbox pane.

Programming for many purposes

The Scratch player applet

Scratch projects are normally executed within the Scratch development environment shown in [Image 6](#). However, you must have the Scratch software installed on your computer to execute and view the projects that way. The animations that you will see in this module are produced by a Java applet that is created by MIT to cause the scratcher's shared projects to be accessible online with no requirement for the viewer to have the Scratch software installed.

These applets usually start running as soon as the web page that contains them is loaded into the browser. *(Sometimes you need to click the green flag in the upper-right corner to cause the applet to start running.)* You can stop the execution of an applet by clicking the red button in the upper-right corner. You can restart the execution of the applet by clicking the green flag.

Computer art by pandalecteur

The scratchers who create and post these projects come at it from many different viewpoints, but they are all programming in order to accomplish their own purposes. My guess is that in many cases, programming is simply a means to an end. For example, I would categorize the scratcher named **pandalecteur**, whose work you will see in [Image 1](#), as a serious artist. You will find a link to more of her creations [here](#). She combines her programming talent with her artistic talent to produce beautiful animated creations.

Image 1. Rushes with sun by pandalecteur.

Figure

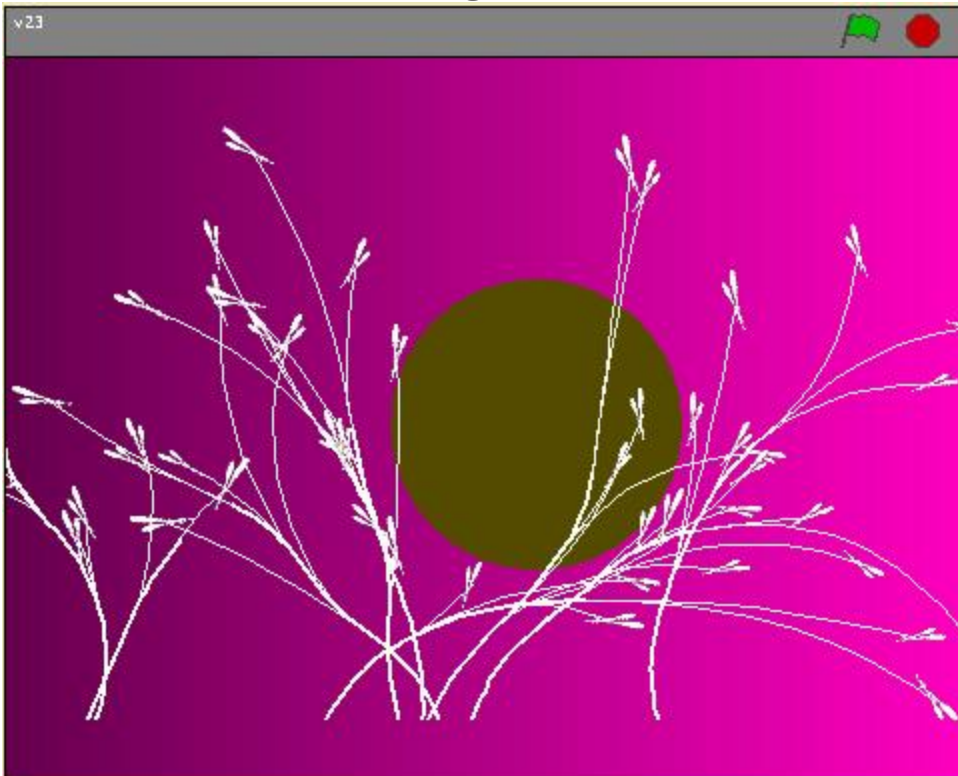


Image 1. Rushes with sun by pandalecteur.

Click [here](#) to view the applet. Close the applet page when finished viewing.

Click the link to run the applet

[Image 1](#) contains a screen shot from the applet along with a link that you can click to open another page (*window*) and cause the applet to run in that page.

Be sure to close the pages

Be sure to close the pages containing the applets when you finish viewing them (*or at least click the red button*) to avoid having multiple applets running at the same time as you read the remainder of this module.

What if the applets don't run?

You may find that these applets won't run on your computer. If so, that probably means that the Java runtime environment is not installed on your computer. You can download and install the Java runtime software by clicking [here](#).

Anime projects

[Image 2](#) represents another category of Scratch projects commonly seen on the Scratch web site. Many of the scratchers create the characters for what amounts to animated comic strips often referred to as *anime*. Sometimes the characters are used to tell a story and sometimes the programmers/authors/artists are content to simply draw and display the characters. Sometimes I read about two scratchers getting together with one doing the art work and the other doing the animation programming.

Image 2. How to draw a head by dialga.

Figure



Image 2. How to draw a head by dialga.

Click [here](#) to view the applet. Close the page when finished viewing.

[Image 2](#) shows an educational animation with instructions on how to draw the head of an anime character. If you run the applet and click on the green button labeled **next** in the upper-left corner several times in succession, the program will cycle through the steps involved in drawing a head.

I described **pandalecteur** earlier as a *serious artist* . In the same vein, I would describe **dialga** who created the project shown in [Image 2](#) as an *anime artist* . I'm not discrediting or indicating a preference for either artist. I am simply recognizing the different viewpoints of the two. Click [here](#) to view more of **dialga's** creations.

Game projects

As you may have guessed, many scratchers like to create games. Most of them are 2D games such as the Marble Racer game by **jamie** shown in [Image 3](#) where the object of the game is to use the arrow keys to cause the marble to roll all the way around the track without being slowed down by rolling onto the grass. Click [here](#) to see more projects by **jamie** .

In this case, you need to click the green flag in the upper-right corner to start or re-start the game. Click the red button to stop the game.

Image 3. Marble Racer SBE 4000 by jamie.

Figure



Image 3. Marble Racer SBE 4000 by jamie.

Click [here](#) to view the applet. Close the page when finished viewing.

3D games

Although the mathematics involved are probably too advanced for most scratchers, a few scratchers push the envelope and attempt to make 3D games. My observation so far is that most of the 3D games that they make are so slow as to be extremely boring. Nonetheless, working on a 3D game in Scratch is a very educational process since Scratch doesn't provide any built-in support for doing 3D projections other than providing the necessary trigonometric functions.

A pseudo-3D animation

[Image 4](#) shows a pseudo-3D animation of my own design. In this project, the walking sprite follows the mouse pointer. Various tricks were employed to create an illusion of perspective in order to create an illusion of 3D. If the sprite doesn't walk fairly smoothly, make sure that you have stopped the execution of all of the other applets.

Image 4. Perspective03 by dbal.

Figure



Image 4. Perspective03 by dbal.

Click [here](#) to view the applet. Close the page when finished viewing.

An animated story

[Image 5](#) shows an animated story named **Day Dream** by **cremeglace** . In this project, the main character has a dream that goes through several scenes before reaching the end of the dream. The project includes music and dancing sprites.

Image 5. Day Dream by cremeglance.

Figure



Image 5. Day Dream by cremeglance.

Click [here](#) to view the applet. Close the page when finished viewing.

The tip of the iceberg

These five example projects are simply the tip of the iceberg in terms of the different purposes for which scratchers create projects. If you can imagine it in an animated computer program, some scratcher has probably [already tried to do it](#).

Scratch development environment

[Image 6](#) shows a reduced screen shot of the Scratch development environment. The text in [Image 6](#) is too small to read in most cases. The main purpose of [Image 6](#) is to show you the overall layout of the Scratch development environment.

Image 6. Screen shot of the Scratch development environment.

Figure

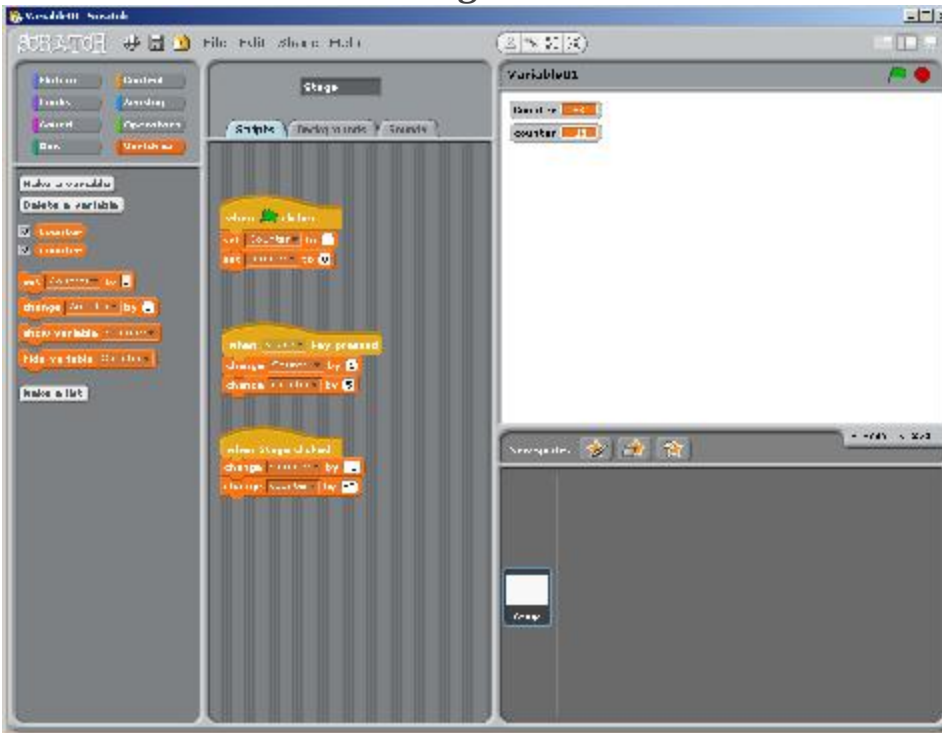


Image 6. Screen shot of the Scratch development environment.

A sprite-oriented programming environment

Scratch is a *sprite-oriented* programming environment. Every Scratch project consists of none, one, or more sprites and a stage upon which the sprites perform the behavior for which they are designed.

A sprite is an invisible entity to which the programmer can assign numerous behaviors. Each sprite can have one or more costumes. The assignment of a visible costume to an invisible sprite causes the sprite to also become visible.

Costumes

The Scratch development environment provides a large gallery of costumes and you can create your own costumes if you don't find any in the gallery that suit your needs. A costume is nothing more than an image. The background in the image is normally transparent. You can import images from a variety of different file types to create costumes. Probably the most difficult task in creating a costume from an imported image is causing the background to be transparent (*if the image didn't originally have a transparent background*). A built-in paint program is provided to assist in this task. You can also use programs such as the free [GIMP](#) image editor to deal with the background.

Walking-boy costumes

The project shown in [Image 4](#) uses five costumes from a family of seven costumes that are available in the gallery. Six of the seven costumes are shown in [Image 7](#). The project shown in [Image 4](#) uses only the five rightmost costumes shown in [Image 7](#).

Image 7. Costumes for a boy walking.

Figure



Image 7. Costumes for a boy walking.

As you can see, the five rightmost costumes describe five stages in a boy walking. The project shown in [Image 4](#) cycles through the five costumes in such a way as to make it appear that the boy is walking. (*This process is often called frame animation and dates back to or before the earliest days of the Disney productions.*)

The stage

The stage is the large white rectangle in [Image 6](#). Different images can be assigned to the stage to serve as backgrounds for the performance of the sprites. The gallery contains a large number of background images, and you can also create your own. Usually for a background image, you don't have the transparency issue discussed earlier so backgrounds for the stage can be easier to create than costumes for sprites.

A moonscape background from the gallery was used for the project shown in [Image 4](#).

Controlling the behavior of a sprite

You control the behavior of a sprite by writing one or more scripts that belong to the sprite. In reality, each script is an event handler of sorts. Usually you will have one event handler that executes when the user clicks the green flag at the top right in [Image 6](#). This is particularly true if any special initialization code is required for the sprite, such as setting the current costume to a particular costume, for example.

In addition to the green-flag script, you can also write scripts to handle mouse events, key events, and a special type of event that fires when one sprite broadcasts a message to one or more other sprites. Therefore, Scratch projects are not only sprite-based; they are also event-based. These are relatively advanced concepts in most programming environments. However, Scratch was designed in such a way as to make it easy to write event-handler scripts to control the behavior of sprites.

Drag and drop programming

Creating Scratch programs involves very little typing. Instead, scripts are created by constructing stacks of blocks, where each block imparts some specific behavior to a sprite or to the stage. For example, the project being developed in [Image 6](#) has a stage, (*which is always the case*) but doesn't yet have any sprites. The stage is represented by the rectangular thumbnail image in the bottom right area of [Image 6](#). Any sprites that are added to the project would also appear as thumbnail images in that same area. I will refer to this area as the sprite list area.

The physical process for writing a script is as follows:

1. Select the stage or a sprite in the sprite list area that is to be programmed.
2. Select the Scripts tab in the center pane.
3. Select one of the color-coded buttons in the toolbox button area at the top left to expose the blocks contained in a particular toolbox in the toolbox pane. The toolbox pane is the large pane on the left.
4. Drag blocks from the toolbox to the scripts (*center*) pane. (*Several blocks have already been dragged to the scripts pane in [Image 6](#).*)

5. Go back to 3 and continue this process until you have all of the blocks that you need in the scripts pane.
6. Snap the blocks together in the correct arrangement to create a script that produces the desired behavior.

You will learn the details of such operations in future modules.

Costumes and sounds

Selection of the other two tabs showing at the top of the center pane in [Image 6](#) exposes the controls for importing, editing, and creating costumes and backgrounds, as well as recording and/or importing sound files to be used as music and sound effects.

The Scratch programming language

Despite the fact that Scratch has amassed a huge following since its Beta release on March 4, 2007, in my opinion, the language isn't a particularly good language from a computer science viewpoint. Of the ten or fifteen programming concepts that most computer science professors consider to be fundamental to good programming, Scratch supports only a few. Those few include:

- Variables and literals
- Sequence, selection, and loop structures
- Arithmetic, relational, and logical operators

These are very important concepts but notably absent is programmer-defined functions or methods.

However, what Scratch lacks in the support of fundamental programming concepts, it makes up for in high-level multimedia functionality. In effect, Scratch provides just enough in the way of fundamental programming capability to form a bridge to high-level multimedia functionality.

Music, music, music

Among other things, *(particularly including the availability of a quasi social-networking community)* , it is probably this multimedia functionality that has attracted the large following among middle school and high school students.

There are a few projects on the website containing original music developed by the scratcher. *(I recall seeing one project where the scratcher was offering to compose original music to be used by other scratchers.)* However, that is the exception and not the rule. Most of the music on the website appears to be copyright music for which the scratcher probably doesn't own the copyright.

That having been said, Scratch provides the capability for scratchers to compose their own music if they choose to do so.

The toolbox buttons and toolbox pane

[Image 8](#) shows a near normal size screen shot of the leftmost portion of [Image 6](#).

Image 8. The toolbox buttons and toolbox pane.

Figure

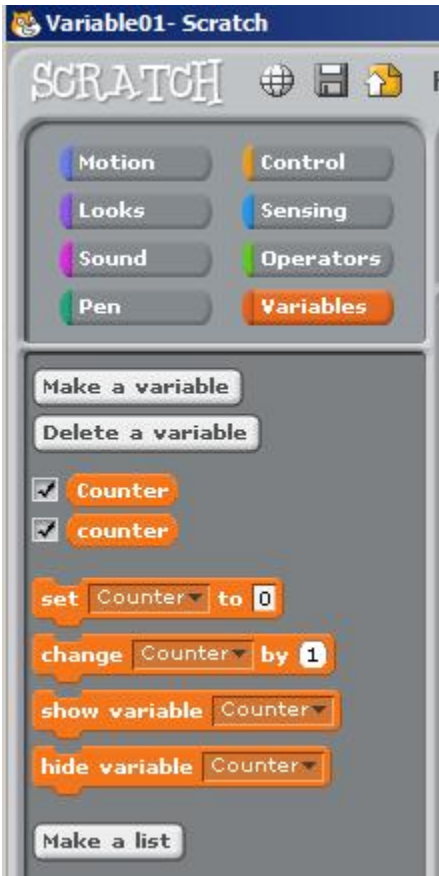


Image 8. The toolbox buttons and toolbox pane.

The small frame at the top of [Image 8](#) show the eight toolbox buttons that you click to expose a corresponding toolbox of blocks in the lower frame. (The **Variables** toolbox was selected before capturing the screenshot shown in [Image 8](#).)

High-level multimedia functionality

With regard to high-level multimedia functionality, Scratch makes it possible and relatively easy for scratchers to do the things they like to do such as:

- Create animation using blocks from the **Motion** toolbox.
- Deal with costumes, colors, graphics effects and sprite size using blocks from the **Looks** toolbox.
- Play imported music and sound effects and create original music and sound effects using blocks from the **Sound** toolbox.
- Create program-controlled line drawings using blocks from the **Pen** toolbox.

These capabilities, which are often difficult to access using "*typical CS-approved*" programming languages, provide a great deal of sensory feedback and they are a lot of fun to program. This is a large part of what causes Scratch to engage aspiring programmers.

Program control

This high-level multimedia functionality is controlled by:

- Programming event firing and detection, loop structures, and selection structures using blocks from the **Control** toolbox.
- Obtaining a variety of different types of information to be used in decision structures using blocks from the **Sensing** toolbox.
- Performing arithmetic, relational, and logical operations and accessing a variety of math functions using blocks from the **Operators** toolbox.
- Creating and servicing variables using blocks from the **Variables** toolbox.

As a practical matter, the four buttons on the left in [Image 8](#) are the *fun* buttons and the four buttons on the right are the *control* buttons.

Summary

In this module, I have provided a high-level overview of the [Scratch](#) programming environment. Future modules will provide more detailed information about the programming environment.

Resources

- [Scratch home](#)
- [Scratch download page](#)
- [Scratch tutorial - Dance Tutorial](#)
- [Scratch forums](#)
- [Son of String Art](#)
- [Scratch explanatory video](#)
- [Scratch tutorials from MIT](#)
- [Scratch tutorial - Digital Art for All](#)
- [Scratch for Budding Computer Scientists](#)
- [Learn Scratch at learnscratch.org](#)
- [Scratch Tutorial - Space Shuttle Mission STS-2020](#)
- [SCRATCH TUTORIALS ICT In Primary Education 2012-2013](#)
- [Scratch Wiki](#)
- [Scratch WikiTable of ContentsWebsite - Scratch Wiki](#)
- [Scratch WikiTable of ContentsTutorials - Scratch Wiki](#)
- [Scratch WikiTable of ContentsProgram - Scratch Wiki](#)
- [Tutorial about Variables - Scratch Wiki](#)
- [Scratch support MIT web site](#)
- [Scratch resources at Scratch.ie](#)
- [ScratchEd Resources](#)
- [Scratch project Day Dream](#)
- [Scratch project Son of String Art](#)
- [Scratch Project Scratch Tutorial](#)

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Scr0100: Overview
- File: Scr0100.htm
- Published: 03/15/13

- Revised: 03/24/13

Note: Disclaimers:

Financial : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.

I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module. In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.

Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Scr0110: Getting Started

The purpose of this module is to get you started programming in Scratch.

Table of Contents

- [Preface](#)
 - [Let's have some fun in the process of learning](#)
 - [Fully interactive operating mode](#)
- [Setup your computer to use Scratch](#)
 - [Run the tutorials](#)
 - [Programming fundamentals](#)
 - [Will concentrate on the fundamentals](#)
 - [Drag and drop programming](#)
 - [Mechanics are difficult to explain](#)
 - [Advantages and disadvantages](#)
 - [Run some student projects](#)
 - [View the Help menu in the Scratch development environment](#)
 - [Selecting Help Page...](#)
 - [Selecting Help Screens...](#)
- [What's next?](#)
- [Resources](#)
- [Miscellaneous](#)

Preface

This module is one in a collection of modules designed to help beginners of all ages (*8 and up*) learn how to create the code for computer programs. Information is provided not only for the beginners themselves but also for their parents and teachers where appropriate.

The purpose of this module is to get you started programming in [Scratch](#).

Let's have some fun in the process of learning

Scratch was designed not only to be a good learning resource, it was also designed to make learning to program fun. After all, if you and/or your students are going to invest the time to learn how to write computer programs, you might as well have a little fun along the way. I suggest that you go to the [Scratch home page](#) and click on a few projects to see what I mean. Just keep in mind that many of the projects that you will find there were written by youngsters and may or may not work as advertised.

One of my favorite Scratch projects is named [Son of String Art](#) (*hopefully it is still there*). You should be able to click on the little yellow shapes in the green buttons at the top of the viewing area and see your shape constructed with strings resembling the string art of the sixties.

Fully interactive operating mode

Note the use of the word *interactive* in the above header. What you will see when you select a Scratch project on the MIT website isn't simply a passive embedded video like you might see at [YouTube](#) or [Vimeo](#). Instead it is a fully interactive program that allows you to control the program by clicking buttons, etc., provided of course that the program was written to be used in an interactive mode.

Setup your computer to use Scratch

The first programming language that I will use to help you learn about computer programming is [Scratch](#). (*I explained quite a bit about Scratch in the earlier module titled [Scr0100: Scratch Overview](#).*) Therefore, the first thing that you need to do is to get your computer setup to run Scratch. Go to the Scratch download page (see [Resources](#)) to download and install the free Scratch software.

I am assuming that you already know how to download and install software on your computer. I installed Scratch version 1.4 on a Windows 7 machine a few days ago. I also reinstalled Scratch on an older Windows XP machine in order to upgrade to version 1.4. Everything went smoothly with no difficulties in either case. *(I haven't tried Scratch with Windows 8 as of this writing but several people on the Scratch forum are reporting "no problem" with Scratch on Windows 8..)*

If you don't know how to download and install the Scratch software, perhaps you can find a friend, relative, or neighbor who can assist in that regard.

Run the tutorials

Once you have Scratch installed on your computer, I recommend that you run every Scratch tutorial that you can find beginning with the tutorials at http://info.scratch.mit.edu/Video_Tutorials.

I have listed several tutorials in [Resources](#). Some are pretty good and some are not so good, but regardless, it would probably be useful for you to work through them all if you have the time available.

I also recommend that you run and pay particular attention to the [Scratch explanatory video](#) from MIT.

Programming fundamentals

I doubt that you will find many Scratch tutorials that cover the fundamental programming concepts that I plan to cover in these modules. Most of the available Scratch tutorials tend to ignore the fundamentals and jump immediately into the higher-level features of Scratch such as playing music or viewing an image through a fisheye lens. There's nothing wrong with that. It is those features that are likely to keep today's students engaged long enough to learn the fundamentals and go on to bigger and better things.

Will concentrate on the fundamentals

However, the modules in this collection will concentrate on the fundamental concepts of computer programming as identified in the earlier module titled [Tkc0100: Preface](#).

The fundamental concepts are transferable from one programming language to the next, and are required for learning the more advanced aspects of computer science.

The higher-level features of Scratch are not necessarily transferable to other languages. What I mean by that is that just because you know how to compose and play music in Scratch doesn't mean that you know how to compose and play music in some other programming language. However, if you understand the fundamental programming concepts embodied in Scratch, you should be able to transfer that understanding to other programming languages.

Drag and drop programming

One of the reasons that it will be useful for you to work through the tutorials is to help you gain proficiency in the *drag and drop* programming paradigm used with Scratch. This is a relatively new programming paradigm, and it is very different from the paradigm used for *old school* languages such as C++, C#, and my favorite programming language, Java.

"It seems strange to speak of Java as an old school programming language, but it has been a viable programming language for a little more than fifteen years now. In today's fast paced technology world, that probably makes it old school."

Mechanics are difficult to explain

Although drag and drop programming has some major advantages over the old paradigm for the beginning programming student, the mechanics

involved are much more difficult to explain in a written tutorial than the mechanics of programming with Java, for example.

With the old school paradigm, all that is necessary to create a program is to:

- Open a text editor and type in the program.
- Open a compiler and compile the program.
- Execute the program.

Advantages and disadvantages

Because everything in old school programming is based on text, it is easy to explain it in a text-based written tutorial. The disadvantage of old school programming insofar as the student is concerned is that the student is required to memorize difficult and sometimes arcane programming syntax, which is often the downfall of many aspiring programming students.

The advantage of drag and drop programming is that the student is not required to memorize programming syntax; at least not in the beginning. That leaves the student free to concentrate on concepts.

One disadvantage of drag and drop programming is that the mechanics are very difficult to explain in a written tutorial. It is sort of like trying to explain to someone how to tie their shoes in a written tutorial. You almost need to see it done in order to learn how to do it yourself.

The best way to explain many aspects of drag and drop programming is through "*show me*" videos.

Many of the available tutorials are video tutorials, which make it possible for you to see how others accomplish drag and drop programming using Scratch. Once you see how it is done, you should have no difficulty doing it yourself.

Run some student projects

As I mentioned earlier, I also recommend that you run some of the [student projects](#) just to see what students are doing with Scratch. As I mentioned earlier, I believe that the projects are delivered to your browser as Java applets (*or something similar*) and most modern browsers are capable of running applets straight out of the box. If you do encounter problems running the student projects, you might consider seeking help on the Scratch forums (see [Resources](#)) .

View the Help menu in the Scratch development environment

In an earlier module, I showed you a [screen shot of the Scratch development environment](#) . Once you have installed the Scratch development environment, you should take a look at the **Help** material that is available locally in the Scratch development environment on your computer.

If you pull down the Help menu, you should see at least the following two items:

- Help Page
- Help Screens

Selecting Help Page...

As of March 2013, selecting **Help Page** from that menu will open an HTML file in your browser containing at least the following four items:

1. Getting Started
2. Help Screens
3. Reference Guides
4. Visit the Scratch support page

As the name implies, selecting **Getting Started** will open a PDF file containing information to help you get started coding in Scratch.

Selecting **Help Screens** will open an HTML file in your browser that explains the behavior of all, or at least most of the programming blocks that are available to the Scratch programmer.

Selecting **Reference Guides** will open a PDF file containing technical reference information about Scratch.

Selecting **Visit the Scratch support page** will open a page on the [MIT web site](#) containing a variety of information about programming with Scratch.

Selecting Help Screens ...

Selecting the **Help Screens...** menu item in the Scratch development environment will open the same HTML file in your browser mentioned [above](#) that explains the behavior of all, or at least most of the programming blocks that are available to the Scratch programmer.

What's next?

In the next module, I will begin the process of helping you to learn about the following computer programming concepts using Scratch:

- Memory
- Variables
- Literals

Resources

- [Scratch home](#)
- [Scratch download page](#)
- [Scratch tutorial - Dance Tutorial](#)
- [Scratch forums](#)
- [Son of String Art](#)
- [Scratch explanatory video](#)
- [Scratch tutorials from MIT](#)
- [Scratch tutorial - Digital Art for All](#)

- [Scratch for Budding Computer Scientists](#)
- [Learn Scratch at learnscratch.org](#)
- [Scratch Tutorial - Space Shuttle Mission STS-2020](#)
- [SCRATCH TUTORIALS ICT In Primary Education 2012-2013](#)
- [Scratch Wiki](#)
- [Scratch WikiTable of ContentsWebsite - Scratch Wiki](#)
- [Scratch WikiTable of ContentsTutorials - Scratch Wiki](#)
- [Scratch WikiTable of ContentsProgram - Scratch Wiki](#)
- [Tutorial about Variables - Scratch Wiki](#)
- [Scratch support MIT web site](#)
- [Scratch resources at Scratch.ie](#)
- [ScratchEd Resources](#)
- [Scratch project Day Dream](#)
- [Scratch project Son of String Art](#)
- [Scratch Project Scratch Tutorial](#)

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Scr0110: Getting Started
- File: Scr0110.htm
- Published: 03/17/13
- Revised: 03/30/13

Note: Disclaimers:

Financial : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you

should be aware that some of the HTML elements in this module may not translate well into PDF.

I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module. In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.

Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Scr0120: Memory, Variables, and Literals

The purpose of this module is to teach you about memory, variables, literals, and algorithms. You will also learn how to write a Scratch program that illustrates the creation and use of variables, case sensitivity, and the ability to detect and respond to mouse and keyboard events.

Table of Contents

- [Preface](#)
 - [Student programming projects](#)
 - [Viewing tip](#)
 - [Images](#)
- [General background information](#)
 - [An overview of computer programming](#)
 - [Not especially difficult](#)
 - [One step at a time](#)
 - [The hard work is often done for us](#)
 - [Memory](#)
 - [Random access versus sequential memory](#)
 - [Combination random/sequential memory](#)
 - [Variables](#)
 - [Brief definition of a variable](#)
 - [A physical analogy](#)
 - [Pretend that you are a computer program](#)
 - [Names versus addresses](#)
 - [Execute an algorithm](#)
 - [Declaring a variable](#)
 - [Literals](#)

- [Where does data originate?](#)
 - [Hard coded data](#)
 - [A value coded into the program](#)
- [Preview of the Scratch program](#)
 - [The Scratch program named Variable01.sb](#)
 - [A variable named Counter](#)
 - [A variable named counter](#)
- [Discussion and sample code](#)
 - [No sprites](#)
 - [The toolbox buttons](#)
 - [A full-size view of the Variables area](#)
 - [Create two variables](#)
 - [Allowable operations for variables](#)
 - [Specifying the values](#)
 - [Cause the variables to be displayed](#)
 - [Create a new variable](#)
 - [Allowable variable names and lengths](#)
 - [Meaningful variable names](#)
 - [Writing the program](#)
 - [Programming blocks in the Control category](#)
 - [The finished program](#)
 - [Behavior of the program](#)
 - [Click the green flag](#)
 - [Press the space bar](#)
 - [Click the mouse on the Stage](#)
 - [Event-driven programming](#)
 - [Example blocks with pointed ends](#)

- [Run the program](#)
- [Student programming projects](#)
 - [Project 1](#)
 - [Project 2](#)
- [Summary](#)
- [What's next?](#)
- [Resources](#)
- [Miscellaneous](#)

Preface

This module is one in a collection of modules designed to help beginners of all ages (*8 and up*) learn how to create the code for computer programs. Information is provided not only for the beginners themselves but also for their parents and teachers where appropriate.

The purpose of this module is to teach you about memory, variables, literals, and algorithms. You will also learn how to write a [Scratch](#) program that illustrates the creation and use of variables, case sensitivity, and the ability to detect and respond to mouse and keyboard events.

Student programming projects

In addition to presenting and explaining a Scratch programming project, I will present two student programming projects that are designed to:

- Help the student retain the knowledge gained by studying the module.
- Require the student to think beyond the material presented in the module by requiring the student to answer the question "How do I ...?"

Viewing tip

I recommend that you open another copy of this module in a separate browser window and use the following links to easily find and view the images while you are reading about them.

Images

- [Image 1](#). A reduced image of the Scratch user interface for the program named Variable01.
- [Image 2](#). A view of the Variables area of the Scratch user interface.
- [Image 3](#). Dialog for entering a variable name.
- [Image 4](#). Programming blocks in the Control category.
- [Image 5](#). The finished program.
- [Image 6](#). Example blocks with pointed ends.
- [Image 7](#). Output from Project 1.
- [Image 8](#). Output from Project 1.
- [Image 9](#). Output from Project 2.
- [Image 10](#). Output from Project 2.

General background information

An overview of computer programming

I recognize that some of you may not be ready for the kind of technical detail that I will provide in this section. If so, feel free to skip ahead to the section titled [Preview of the Scratch program](#). Just remember that this material is here waiting for you to come back and study it when you are ready.

Not especially difficult

Computer programming is not especially difficult. However, it does require an aptitude for solving problems. In fact, a computer program is usually a model for a solution to someone's problem.

One step at a time

If you can learn to program a Digital Video Recorder (DVR), you can probably also learn to program a computer. Of course, a computer is more complicated than a DVR, so there is more to learn. The important thing is to take the learning process in deliberate steps making certain that you understand each step before moving on to the next one.

The hard work is often done for us

Fortunately, when we write programs using a high-level programming language such as Scratch, much of the hard work is done for us behind the scenes. For example, we don't have to perform the small incremental steps that are required to divide one number by another number.

As programmers, we are more like conductors than musicians. The various parts of the computer represent the musicians. We tell them what to play, and when to play it, and if we do our job well, we produce a solution to someone's problem.

Memory

All computers contain *memory* of one type or another. When we speak of human memory, we are usually speaking of the things that the human remembers. However, when we speak of computer memory, we are usually speaking of physical devices where data is stored for later retrieval. Most modern computers contain memory of a type that is often referred to as RAM (*more on RAM later*) .

All data is stored in a computer as numeric values. Computer programs do what they do by executing a series of *operations* on the numeric data.

Numeric data Even the text in this module is stored as numeric data in your computer. For example, the upper-case character A

is commonly represented by the numeric value 65.

The operations that are performed on the numeric data generally consist of calculations and comparisons. It is the order and the pattern of those operations that distinguishes one computer program from another.

Random access versus sequential memory

The reason the memory in most modern computers is called *random access memory (RAM)* is that it can be accessed in any order. Some types of memory, such as a magnetic tape, can only be accessed in sequential order. *(Yes, I did use computers with magnetic tape memory in my younger days during the sixties and seventies.)*

Sequential access means that to get a piece of data that is stored deep inside the memory, it is necessary to start at the beginning and examine every piece of data until the correct one is found. This is typically a slow process.

Combination random/sequential memory

Other types of memory, such as disks provide a combination of sequential and random access. For example, the data on a disk is stored in tracks that form concentric circles on the disk. The individual tracks can be accessed in random order, but the data within a track must be accessed sequentially starting at a specific point on the track. While faster than magnetic tape, even this process is usually slower than true random access memory.

Sequential memory or combination random/sequential memory isn't very useful for most computer programs at the inner working level because access to each particular piece of data is slow. As a result, most modern computers have random access memory that is used for storage and retrieval of data by programs while they are running and disks that are used

for long-term storage and retrieval of data that needs to be saved over longer periods of time.

Variables

As the computer program performs its operations in the prescribed order, it is often necessary for it to store intermediate results somewhere in its memory and to retrieve those results later for use in subsequent operations. The intermediate results are often stored in little chunks of memory that we refer to as *variables* .

Brief definition of a variable

The following definition is paraphrased from [Wikipedia](#).

*In computer programming, a **variable** is a storage location and an associated symbolic name (an identifier) that contains some known or unknown quantity or information -- a value.*

The variable name is the usual way to reference the stored value; this separation of name and content allows the name to be used independently of the exact information it represents.

The value stored in the variable may change during program execution.

A physical analogy

We can think of random access memory as being analogous to a metal rack containing a large number of compartments. The compartments are all the same size and are arranged in a regular grid of rows and columns.

Each compartment has a numeric address printed above it. No two compartments have the same numeric address. Each compartment also has a little slot into which you can insert a name or a label for the compartment. No two compartments can have the same name.

Although the analogy is not perfect, we can think of one of those compartments as being analogous to a variable.

Pretend that you are a computer program

Think of yourself as a computer program. You have the ability to create labels for each compartment (*variable*). You have the ability to write values on little slips of paper and to put them into the compartments. You also have the ability to read the values written on the little slips of paper and to use those values for some purpose. However, there are four rules that you must observe:

- You may not remove a slip of paper from a compartment without replacing it by another slip of paper on which you have written a value.
- You may not put a slip of paper in a compartment without removing the one already there.
- You may not give the same label to two or more compartments.
- You may not change a label once you have assigned it to a compartment.

Names versus addresses

Although each *compartment* in the physical memory in the computer has a numeric address, as a programmer using a *high-level* programming language such as Scratch, you usually don't need to be concerned about the numeric addresses of the compartments. (*The compartments are often referred to as locations in memory.*)

Instead, you can think about the compartments and refer to them in terms of their names. (*Names are easier for most people to remember and understand than numeric addresses.*) Keep in mind, however, that computer memory locations don't really have names. Deep inside the computer program, the names that you use to identify compartments in memory are cross-referenced to memory addresses. At the lowest level, the program works with memory addresses instead of names.

Execute an algorithm

A computer program always executes some sort of procedure or algorithm. The algorithm may be very simple (*such as how to make a peanut butter sandwich*) , or it may be very complex as would be the case for a spreadsheet program. As the program executes the algorithm, it uses the random access memory to store and retrieve the data that is needed to execute the algorithm.

Declaring a variable

In Scratch, it is necessary to establish the *name* of a variable before you can use it. (*That is not the case in some programming languages such as Python.*)

In programmer jargon, this is referred to as *declaring a variable* . The process of declaring a variable causes memory to be set aside to contain a value, and causes that chunk of memory to be given a name. That name can be used later to refer to the value stored in that chunk of memory.

Some programming languages, (not including Scratch), require that the variable declaration also specify the type of data that will be stored in a variable.

Case sensitivity

The name of a variable is case sensitive in Scratch (*but the name may not be case sensitive in other programming languages*) . In other words, the name **av**ariable is not the same as the name **a**Variable in case-sensitive programming languages. If both names were used in the same program, they would refer to two different variables.

Use names to access values

Variables have names like **price** , **cost** , and **markup** . The names of the variables give us easy access to the values stored in the variables.

Some operations cause the value of a variable to change while other operations simply access the value without changing it.

Storing and retrieving

As the program executes, the values stored in variables can be replaced by other values. In that way, the values of the variables can change as the program executes. (*In other words, the value of a variable can vary over time.*)

It is also possible to retrieve the value stored in a variable without modifying it. In that way, the values of variables can be used to evaluate expressions without modifying those values.

Literals

Where does data originate ?

The data used by a computer program can originate from a variety of sources. For example, it can be entered from the keyboard. It can be read from a disk file. It can be downloaded from a web site, etc.

Hard coded data

The data values can also be coded into the program when the program is written. In this case, we would call it a *literal* or a *literal value* .

A value coded into the program

That is just about all there is to understanding literals. There are some special rules that come into play in certain situations, and I will discuss those rules at the appropriate points in time. For now, just remember that a *literal* is a value that is coded into the program when the program is written.

Preview of the Scratch program

Now it's time to write and discuss a Scratch program. You should be able to replicate this program.

Hopefully by now you have installed Scratch on your computer and you have worked through all of the tutorials that you can find on Scratch as recommended in the previous module. If not, you should do that before continuing with this module.

In case you skipped the section titled [An overview of computer programming.](#), you still might want to go back and review the section on [Variables](#) .

The Scratch program named Variable01 .sb

This Scratch program illustrates the creation and manipulation of two simple *numeric* variables.

*Prior to the release of Scratch version 1.4, only numeric variables were allowed in Scratch. However, according to the [Scratch 1.4 Release Notes](#), variables can now contain strings of characters. Experimentation indicates that they can also contain the **boolean** values of **True** and **False** (or possibly strings that represent **boolean** values). The previous **Numbers** button has been replaced by an **Operators** button (see [Image 2](#)) because it now contains blocks for manipulating strings.*

This program also illustrates the case sensitivity of variable names along with the ability to detect and respond to events fired by the keyboard and events fired by the mouse.

A variable named Counter

A variable named **Counter** (with an upper-case "C") is created, initialized to 0, and displayed in the upper-left corner of the Stage area in the Scratch user interface (see [Image 1](#)). When the user clicks the green flag in the upper-right corner of [Image 1](#), the value of the variable is set to zero. Each time the user presses the space bar, the value of the variable is increased by 1. Each time the user clicks the mouse in the blank area of the Stage, the value of the variable is decreased by 1.

A variable named counter

Another variable named **counter** (with a lower-case "c") is created, initialized to 0, and displayed below the **Counter** variable on the Stage.

Again, when the user clicks the green flag in the upper-right corner of [Image 1](#), the value of the variable is set to zero. Each time the user presses the space bar, the value of this variable is increased by 5. Each time the user clicks the mouse in the blank area of the Stage, the value of this variable is decreased by 5.

Discussion and sample code

[Image 1](#) shows a greatly reduced image of the Scratch user interface for the program named **Variable01** for Scratch version 1.4.

Image 1. A reduced image of the Scratch user interface for the program named Variable01.

Figure

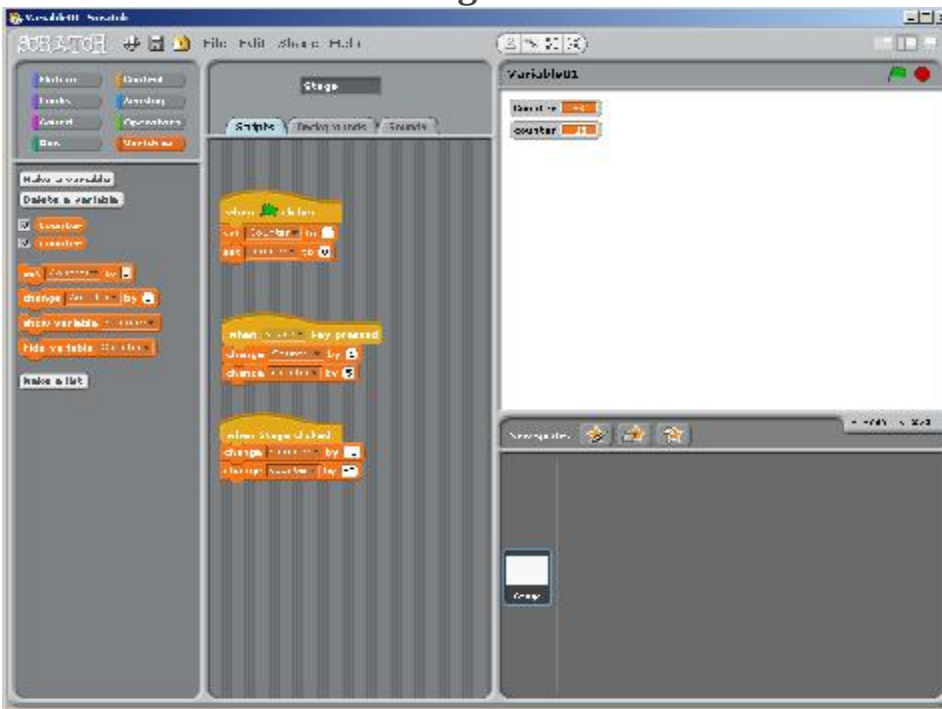


Image 1. A reduced image of the Scratch user interface for the program named Variable01.

You probably can't read any of the text in [Image 1](#). The image is presented here solely to provide an overview of the user interface. I will provide

additional (*more readable*) images later that show and discuss the separate parts of the user interface.

No sprites

There are a few things that we can deduce by viewing the reduced image in [Image 1](#). First, note that there are no sprites in this program. The program consists solely of

- the Stage, shown in white in the upper right,
- a small program consisting of three scripts shown in the middle panel, and
- two variables shown in gray near the top of the lower-left panel and also shown in the upper-left corner of the Stage.

The toolbox buttons

If you have worked through the Scratch tutorials, you will recognize that the material in the lower-left panel is always associated with one of the following eight toolbox buttons **in the upper-left panel**:

- Motion
- Looks
- Sounds
- Pen
- Control
- Sensing
- Operators (*previously Numbers*)
- Variables

In other words, when you click on one of the buttons in the upper-left panel, it exposes a set of tools that you can use to write your program. Those tools appear in the lower-left panel.

When one of those eight buttons is clicked, it becomes completely colored and the tools associated with that button are displayed below it. The orange button has been selected in [Image 1](#). Since you can't read the label on that button, I will tell you that the label on the orange button reads **Variables**.

A full-size view of the Variables area

[Image 2](#) presents a full-size view of the two left-most panels from [Image 1](#) showing the tools that are exposed by the **Variables** button.

***Tools in the Variables toolbox:** When you first click the **Variables** button, only two gray buttons are exposed in the bottom panel of [Image 2](#). One gray button is labeled **Make a variable** and the other gray button is labeled **Make a list**. The orange material in [Image 2](#) was produced by clicking twice on the button labeled **Make a variable** and entering a name for each variable.*

Image 2. A view of the Variables area of the Scratch user interface.

Figure

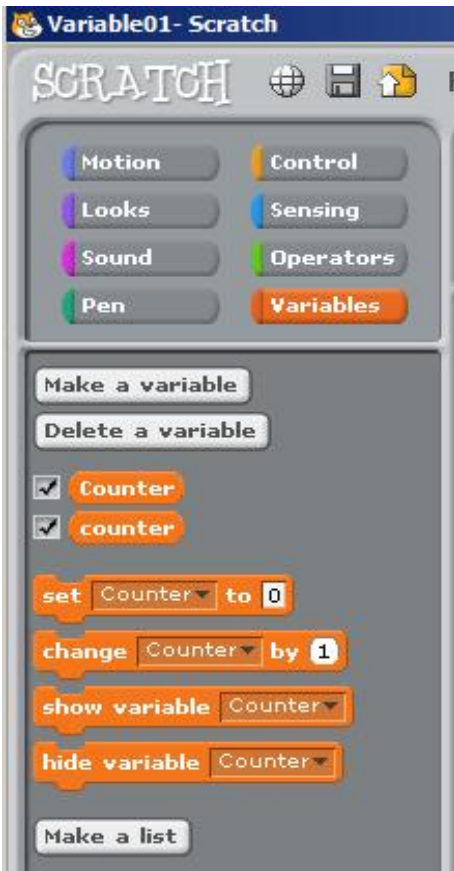


Image 2. A view of the Variables area of the Scratch user interface.

Create two variables

Click [here](#) for instructions on creating variables.

[Image 2](#) shows that this Scratch program contains two variables with the following names:

- **Counter**
- **counter**

I purposely spelled the names of the two variables the same and made them differ only by the case of the first letter to illustrate that Scratch is case sensitive insofar as the names of the variables is concerned. Even though these two variable names contain the same letters, they are two different variables because one name begins with an upper case "C" and the other name begins with a lower-case "c".

Allowable operations for variables

There are four programming blocks (*in the bottom panel of [Image 2](#)*) associated with each variable. As you learned from working through the tutorials, you can drag these blocks into the programming area in the center pane in [Image 1](#) to actually write the program. Thus, there are four different operations that you can perform on a variable:

- You can set the value of the variable to a specified value.
- You can change the value of the variable by a specified value.
- You can show a variable on the stage.
- You can hide a variable.

(You can actually perform the last two operations simply by clicking the block with no requirement to drag it into the center pane. Several operations that required double-clicking prior to v1.4 can now be accomplished with a single click.)

Specifying the values

After dragging a programming block from the Variables area into the programming area, you can:

- Identify the variable to which that block applies by selecting the variable name in the pull-down list built into the block.
- Enter a literal numeric or string value from the keyboard into the white or gray area at the right end of the block, or
- Drag some other block and drop it into the white or gray area.

In some cases, the area looks white and in other cases it looks gray. I will refer to it as the white area in both cases.

There are at least two different shapes that are eligible for being dropped into such areas:

1. A horizontal rectangle with **rounded corners** such as the orange block labeled **Counter** in [Image 2](#). *(This shape seems to always be associated with a numeric or string value.)*
2. A horizontal rectangle with **pointed ends** as shown by several of the blocks in [Image 6](#). *(This shape seems to always be associated with a **boolean** result of true or false.)*

Blocks having the rounded corners are eligible for being dropped into the white areas with rounded corners of the programming blocks in [Image 2](#).

Blocks having the pointed ends are eligible for being dropped into the darker orange areas having a similar shape on the blocks near the bottom of [Image 4](#).

Either shape of block is eligible for being dropped into the white areas with square corners shown in [Image 2](#). Thus the block in [Image 2](#) that is used to set a variable to a value can set the value to a numeric value, a string value, or a **boolean** value.

*(Note that the two variable blocks shown in [Image 2](#) labeled **Counter** and **counter** are eligible for being dropped into the white areas of the two programming blocks shown in [Image 2](#). This makes it possible to cause the value of one variable to depend on the value of another variable.)*

*If you click the **Operators** button shown in [Image 2](#) (see [Image 6](#)) you will expose several different blocks with rounded corners that are eligible for being dropped into the white areas in [Image 2](#) according to the rules described above.*

Cause the variables to be displayed

Checking the boxes to the left of the variable names in [Image 2](#) causes them to be displayed in the upper-left corner of the large white Stage area shown in [Image 1](#). (Clicking the **show** and **hide** blocks will also check and uncheck the checkboxes, which in turn will show or hide the variables on the stage.)

Create a new variable

You create a new variable by clicking the button labeled **Make a variable** shown in [Image 2](#). When you click that button, the dialog shown in [Image 3](#) appears on the screen asking you to provide the name for the new variable.

Image 3. Dialog for entering a variable name.

Figure



Image 3. Dialog for entering a variable name.

At that point, you simply type in the name for the new variable and click the **OK** button.

The radio buttons on that dialog allow you to specify which sprites have access to the variable.

Allowable variable names and lengths

Most programming languages have very specific requirements regarding the allowable characters for variable names. I suspect that this is also true of Scratch, but I haven't found that specification anywhere. You will probably be okay as long as you stick with letters, numbers, the underscore character "_", and the hyphen or minus sign "-".

It appears experimentally that the allowable length of the variable name is longer than you would ever want to use so length doesn't seem to be a limitation.

Many programming languages won't allow you to use a numeric character for the first character in a variable name, but that restriction doesn't seem to apply to Scratch. However, since you might later move up to more mainstream languages, you might want to avoid getting into the habit of using numeric characters as the first character in variable names.

Meaningful variable names

When using Scratch and all other programming languages, you should strive to use variable names that are meaningful. My preference is to begin variable names with a lower-case letter, use multiple words in the name where appropriate, and separate the words using a format commonly called *camelCase* to cause the human eye to separate the words.

(The upper-case characters are analogous to the humps on a camel.)

Here is an example:

aVariableName

Some people prefer the following format:

A_Variable_Name

I prefer the *camelCase* version because it is easier to type, requires fewer characters, and in my opinion, is just as effective.

Writing the program

Generally speaking, programs are written in Scratch by:

1. Selecting buttons in the upper-left of [Image 2](#) to expose the programming blocks associated with each of the categories listed [earlier](#).
2. Dragging programming blocks from the left pane in [Image 1](#) to the center pane in [Image 1](#) and snapping those blocks together in groups to form program scripts.
3. Entering literal numeric values or **dragging other blocks** and dropping them into slots with the same shape to fill out the details of the program.

There are other steps involved in writing a complex program that I didn't include above, but we will get to them later in this collection of modules. Hopefully you already know the physical steps in writing a program as a result of working through the online tutorials.

As you can see in [Image 1](#), this program consists of three scripts in the center pane. Each script contains one tan block and two orange blocks. We already know that orange blocks have to do with variables. As you can see from the colors on the left ends of the buttons in [Image 2](#), tan blocks are related to **Control**.

Programming blocks in the Control category

[Image 4](#) shows some of the programming blocks that are available in the **Control** category. Other programming blocks in this category can be exposed using the vertical slider shown in [Image 4](#).

Image 4. Programming blocks in the Control category.

Figure

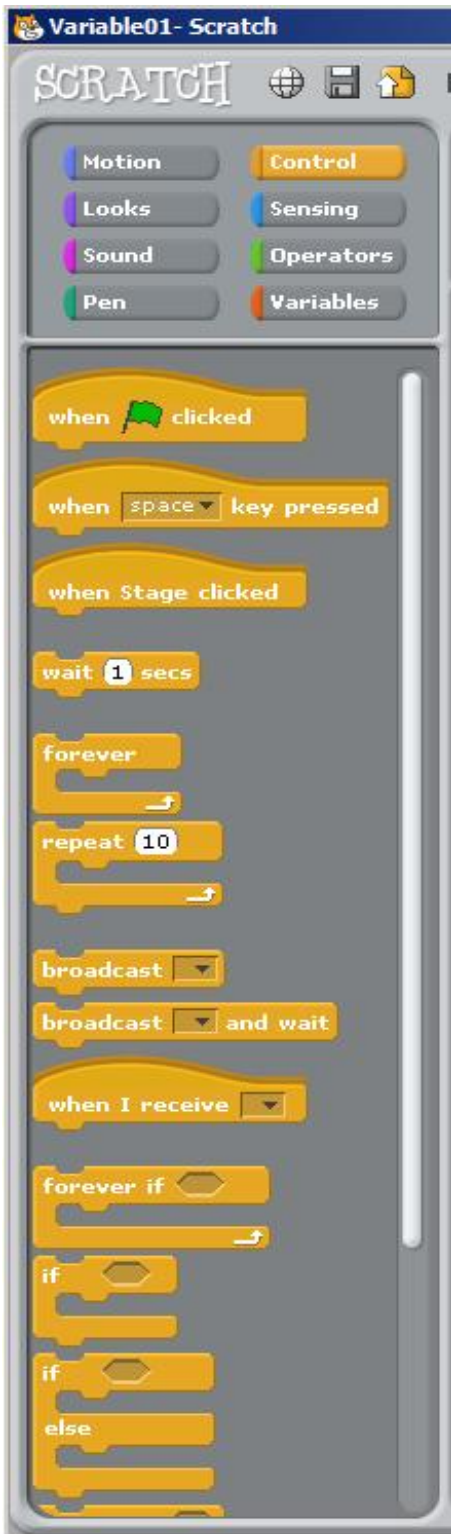


Image 4. Programming blocks in the Control category.

The finished program

[Image 5](#) shows a full-size view of the center panel from [Image 1](#). This image shows the finished program that I created by dragging blocks from the **Variables** area and the **Control** area and snapping them together to form three separate scripts. Note that I also entered literal values of 0, 1, 5, -1, and -5 into the white areas in the programming blocks for the variables after dropping them into the programming area.

In addition, I used the pull-down list on each of six blocks to select the name of the variable to which each block applies. (*Note that each of those six blocks refers either to **Counter** or **counter** .*)

Image 5. The finished program.

Figure

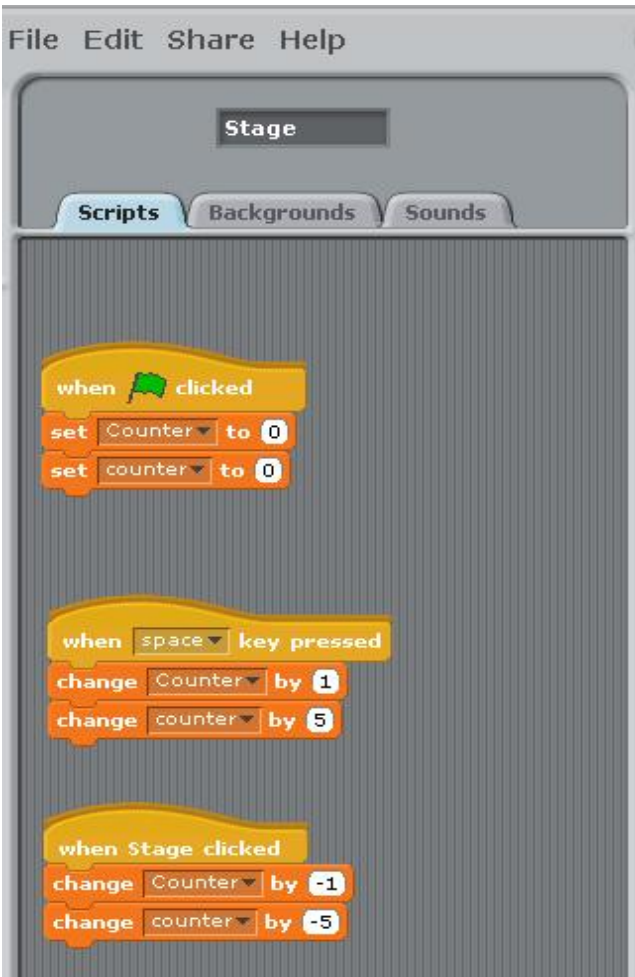


Image 5. The finished program.

Behavior of the program

Click the green flag

The program contains three scripts. Each script remains silent until a specific event occurs. As the label on the uppermost block in the top programming script in [Image 5](#) indicates, the code in the top script in [Image 5](#) is executed once each time the user clicks the green flag shown in the

upper right of [Image 1](#). This code causes the values stored in each of the two variables to be set to zero. This, in turn causes the variable displays in the Stage area to each show a value of 0.

Press the space bar

Because of the top block in the middle script reads *"when space key pressed"* , the code in the middle script is executed once each time the user presses the space bar.

The pull-down list on this block allows you to select among the keys on the keyboard with the space bar being the default.

One of the orange programming blocks in the middle script causes the value of the variable named **Counter** to change by +1 when the space bar is pressed. The other orange programming block causes the value of the variable named **counter** to change by +5 when the space bar is pressed.

In other words, repetitively pressing the space bar causes the **Counter** variable to count up in increments of one and causes the **counter** variable to count up in increments of five.

Click the mouse on the Stage

Because of the top block in the bottom script reads *"when stage clicked"* , the code in the bottom script in [Image 5](#) is executed once each time the user clicks the mouse in the large white Stage area shown in [Image 1](#).

The bottom two blocks in this script are the same as the bottom two blocks in the middle script except that the algebraic signs on the two literal values are minus instead of plus.

The absence of a "-" character causes a literal value to be positive. A "+" character is not required for positive literal values.

Repetitively clicking the mouse in the Stage area causes the **Counter** variable to count down in increments of one and causes the **counter** variable to count down in increments of five.

Event-driven programming

The ability to write programs that cause certain operations to occur in response to events (*such as pressing the space bar and clicking the mouse on the Stage*) is often referred to as *event-driven programming*.

It is easy to write event-driven programs in Scratch. On the other hand, writing event-driven programs is not easy in languages such as Java, C#, and C++. You need quite a lot of programming knowledge to write event-driven programs in those languages.

It is useful to give programming students a taste of event-driven programming early, if for no other reason than the fact that it tends to make programming more interesting.

Example blocks with pointed ends

The blocks shown in [Image 6](#) are not used in this program. They are shown in this module to illustrate blocks with pointed ends referred to [earlier](#).

Image 6. Example blocks with pointed ends.

Figure



Image 6. Example blocks with pointed ends.

Run the program

I encourage you to use the information provided above along with the Scratch Programming Environment shown in [Image 1](#) to write and run this program. Experiment with the code, making changes, and observing the results of your changes. Make certain that you can explain why your changes behave as they do.

A copy of this program has been [posted online](#) for your review.

I also encourage you to write the programs described below.

Student programming projects

The following projects are designed to:

- Help the student retain the knowledge gained by studying the module.
- Require the student to think beyond the material presented in the module by requiring the student to answer the question "How do I ...?"

Project 1

Begin with the program named **Variable01** and modify it to create a new program named **Variable02** . Modify the original program in such a way that pressing the space bar five times in succession, (*after clicking the green flag to set both variables to zero*) , will cause the displayed values of the variables named **Counter** and **counter** to be as shown in [Image 7](#).

Image 7. Output from Project 1.

Figure

Space bar press	Counter value	counter value
1	1	1
2	2	3
3	3	6
4	4	10
5	5	15

Image 7. Output from Project 1.

Having reached that point, clicking the mouse five times in succession in the Stage area will cause the displayed values of the variables named **Counter** and **counter** to be as shown in [Image 8](#).

Image 8. Output from Project 1.

Figure

Mouse click	Counter value	counter value
1	4	19
2	3	22
3	2	24
4	1	25
5	0	25

Image 8. Output from Project 1.

A copy of this program is [posted online](#) for your review.

Project 2

You should successfully complete [Project 1](#) before attempting this project. Begin with the program named **Variable01** and modify it to create a new program named **Variable03** . Modify the original program in such a way that pressing the space bar five times in succession, (*after clicking the green flag to set both variables to zero*) , will cause the displayed values of the variables named **Counter** and **counter** to be as shown in [Image 9](#).

Image 9. Output from Project 2.

Figure

Space bar press	Counter value	counter value
1	1	2
2	2	6
3	3	12
4	4	20
5	5	30

Image 9. Output from Project 2.

Having reached that point, clicking the mouse five times in succession in the Stage area will cause the displayed values of the variables named **Counter** and **counter** to be as shown in [Image 10](#).

Image 10. Output from Project 2.

Figure

Mouse click	Counter value	counter value
1	4	38
2	3	44
3	2	48
4	1	50
5	0	50

Image 10. Output from Project 2.

Hint: Compare the values of the variable named **counter** in [Project 2](#) with the values of the variable named **counter** in [Project 1](#) to determine what you need to do to successfully write [Project 2](#).

A copy of this program is [posted online](#) for your review (see [Resources](#) for the URL) .

Summary

I began by providing an overview of computer programming. Then I provided technical explanations for memory, variables, and literals.

I explained the meaning of the term algorithm.

I presented and explained a Scratch program that is designed to illustrate the creation and use of variables in a computer program. The Scratch program also illustrates case sensitivity in variable names along with the

ability to detect and respond to events fired by the keyboard and events fired by the mouse.

Finally I presented two student programming projects that are designed to:

- Help the student retain the knowledge gained by studying the module.
- Require the student to think beyond the material presented in the module by requiring the student to answer the question "How do I ...?"

What's next?

In the next module, I will continue the process of helping you to learn about the following computer programming concepts using Scratch:

- Variables
- Expressions and Operators
- Sequence, Selection, and Loop Structures

Resources

- [Scratch home](#)
- [Scratch download page](#)
- [Scratch tutorial - Dance Tutorial](#)
- [Scratch forums](#)
- [Son of String Art](#)
- [Scratch explanatory video](#)
- [Scratch tutorials from MIT](#)
- [Scratch tutorial - Digital Art for All](#)
- [Scratch for Budding Computer Scientists](#)
- [Learn Scratch at learnscratch.org](#)
- [Scratch Tutorial - Space Shuttle Mission STS-2020](#)
- [SCRATCH TUTORIALS ICT In Primary Education 2012-2013](#)
- [Scratch Wiki](#)
- [Scratch WikiTable of ContentsWebsite - Scratch Wiki](#)
- [Scratch WikiTable of ContentsTutorials - Scratch Wiki](#)
- [Scratch WikiTable of ContentsProgram - Scratch Wiki](#)
- [Tutorial about Variables - Scratch Wiki](#)

- [Scratch support MIT web site](#)
- [Scratch resources at Scratch.ie](#)
- [ScratchEd Resources](#)
- [Scratch project Day Dream](#)
- [Scratch project Son of String Art](#)
- [Scratch Project Scratch Tutorial](#)
- Online version of program named [Variable01](#)
- Online version of student programming project named [Variable02](#)
- Online version of student programming project named [Variable03](#)

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Scr0120: Memory, Variables, and Literals
- File: Scr0120.htm
- Published: 03/24/13

Note: Disclaimers:

Financial : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.

I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module. In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a

book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.

Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Scr0130: Sequence, Selection, and Loop

The purpose of this module is to teach you about structured programming; the sequence structure, the selection structure, and the loop structure. The module will also teach you how to write a Scratch program that illustrates the selection structure, mouse events, and Cartesian coordinates.

Table of Contents

- [Preface](#)
 - [Viewing tip](#)
 - [Images](#)
- [General background information](#)
 - [What is structured programming?](#)
 - [One door in and one door out](#)
 - [Nesting of structures is allowed](#)
 - [Pseudocode](#)
 - [The sequence structure](#)
 - [The action elements themselves may be structures](#)
 - [The selection structure](#)
 - [Test a condition for true or false](#)
 - [The action elements themselves may be structures](#)
 - [Sometimes no action is required on false](#)
 - [The loop structure](#)
 - [Perform the test and exit on false](#)
 - [Perform some actions and repeat the test on true](#)
 - [Each action element may be another structure](#)
 - [Need to avoid infinite loops](#)

- [Other possible structures](#)
- [Preview](#)
 - [Screen output for the program named IfSimple01](#)
 - [Click the green flag or the basketball](#)
- [Discussion and sample code](#)
 - [Three sprites](#)
 - [Program code for the LeftBeachball](#)
 - [Adding the blue go to block to the program](#)
 - [Coordinate values](#)
 - [Behavior of the LeftBeachball](#)
 - [Positioning the other beach ball](#)
 - [The basketball](#)
 - [Initializing the position of the basketball](#)
 - [Initializing the orientation of the basketball](#)
 - [Handling mouse events on the basketball](#)
 - [Behavior of the bottom script](#)
 - [The selection block](#)
 - [Two independent decisions](#)
 - [Selection control structures available in Scratch](#)
 - [How do the two bottom blocks differ?](#)
 - [Specifying the condition on which the decision will be based](#)
 - [Two groups of programming blocks have the correct shape](#)
 - [Will use the touching block](#)
 - [Go back and examine the script](#)
 - [Conditions have been established - need actions](#)
 - [Programming blocks belonging to the Motion group](#)

- [Turn around and face the other way](#)
- [An online version of this program is available](#)
- [Run the program](#)
- [Student programming project](#)
 - [Orthogonal axes in Cartesian coordinates](#)
 - [Draw a straight line to the location of the next mouse click](#)
 - [A sneak peek at the solution](#)
- [Summary](#)
- [What's next?](#)
- [Resources](#)
- [Miscellaneous](#)

Preface

This module is one in a collection of modules designed to help beginners of all ages (*8 and up*) learn how to create the code for computer programs. Information is provided not only for the beginners themselves but also for their parents and teachers where appropriate.

The purpose of this module is to teach you about structured programming; the sequence structure, the selection structure, and the loop structure. The module will also teach you how to write a Scratch program that illustrates the selection structure, mouse events, and Cartesian coordinates.

The module also presents a student project by which you can demonstrate your understanding of the concepts learned by studying the module.

Viewing tip

I recommend that you open another copy of this module in a separate browser window and use the following links to easily find and view the images while you are reading about them.

Images

- [Image 1](#). The sequence structure in pseudocode.
- [Image 2](#). The selection structure in pseudocode.
- [Image 3](#). The loop structure in pseudocode.
- [Image 4](#). User interface for the program named IfSimple01.
- [Image 5](#). A paraphrased version of the Scratch code.
- [Image 6](#). Program code for the left beach ball
- [Image 7](#). Initializing the position and orientation of the basketball.
- [Image 8](#). All of the code that applies to the basketball.
- [Image 9](#). Pseudocode for a selection structure.
- [Image 10](#). Two independent decisions.
- [Image 11](#). Selection control structures available in Scratch.
- [Image 12](#). Programming blocks belonging to the Sensing group.
- [Image 13](#). Programming blocks belonging to the Motion group.
- [Image 14](#). Initial output from the program named IfWithVar01.
- [Image 15](#). Program output after having clicked twice in the Stage area.

General background information

In this module, I will help you learn about:

- Structured programming.
- The *sequence* structure.
- The *selection* structure.
- The *loop* structure.

In introductory programming courses, you will often hear a lot about something called *structured programming*. In comparison with more modern and complex programming concepts such as *runtime polymorphism*, structured programming is fairly mundane. However, that's not to say that structured programming isn't important. It is very important. But it is just a small bump in the road of learning that leads to a more complete understanding of computer programming, especially object-oriented programming.

What is structured programming ?

Basically, the concept of structured programming says that any programming logic problem can be solved using an appropriate combination of only three programming structures, none of which are complicated. The three structures are known generally as:

- The sequence structure.
- The **selection or decision structure** .
- The loop, repetition, or iteration structure.

One door in and one door out

To understand structured programming, you need to think in terms of a section of program code that has only one entry point and one exit point. It is very important that there cannot be multiple entry points or multiple exit points.

There must be only one way into the section of code and one way out of the section of code.

Nesting of structures is allowed

Another important part of the concept is that structures may be nested inside of other structures provided that every structure meets the basic rules for a structure.

Thus, by nesting simple structures inside of simple structures, large and complex overall structures can be constructed.

Pseudocode

Computer programming source code consists generally (*but not in Scratch*) of programming instructions written in text form with a very specific format

or syntax that is designed to be understood by a computer program. Humans who are not computer programmers might not be expected to understand much of what they see in computer programming source code.

According to [Wikipedia](#),

"The prefix **pseudo** - (from Greek... "lying, false") is used to mark something as false, fraudulent, or pretending to be something it is not."

According to [Wikipedia](#),

"Pseudocode is an informal high-level description of the operating principle of a computer program or other algorithm. It uses the structural conventions of a programming language, but is intended for human reading rather than machine reading. Pseudocode typically omits details that are not essential for human understanding of the algorithm..."

The purpose of using pseudocode is that it is easier for people to understand than conventional programming language code, and that it is an efficient and environment-independent description of the key principles of an algorithm."

The *sequence* structure

We can describe the *sequence* structure using the pseudocode shown in [Image 1](#).

Image 1. The sequence structure in pseudocode.

Figure

```
Enter
  Perform one or more actions in sequence
Exit
```

Image 1. The sequence structure in pseudocode.

Thus, the general requirement for the sequence structure is that one or more actions may be performed in sequence after entry and before exit.

There may not be any branches or loops between the entry and the exit.

All actions must be taken in sequence.

The action elements themselves may be structures

However, it is important to note that one or more of the action elements may themselves be sequence, selection, or loop structures.

If each of the structures that make up the sequence has only one entry point and one exit point, each such structure can be viewed as a single action element in a sequence of actions.

The sequence structure is the simplest of the three, and there's not much more that I can say about it.

The *selection* structure

The *selection* or *decision* structure can be described as shown in the pseudocode in [Image 2](#).

Image 2. The selection structure in pseudocode.

Figure

```
Enter
  Test a condition for true or false
  On true
    Take one or more actions in sequence
  On false
    Take none, one, or more actions in sequence
Exit
```

Image 2. The selection structure in pseudocode.

Test a condition for true or false

Once again, there is only one entry point and one exit point.

The first thing that happens following entry is that some condition is tested for *true* or *false* .

The concept of something being *true* or *false* is commonly referred to as a *boolean condition* in computer programming (*named after George Boole*) .

If the condition is true, one or more actions are taken in sequence and control exits the structure.

If the condition is false, **none** , one or more different actions are taken in sequence and control exits the structure. (*Note the inclusion of the word none here.*)

The action elements themselves may be structures

Once again, each of the action elements in the sequence may be another sequence, selection, or loop structure.

Eventually all of the actions for a chosen branch will be completed in sequence and control will exit the structure.

Sometimes no action is required on false

It is often the case that no action is required when the test returns false. In that case, control simply exits the structure without performing any actions.

The *loop* structure

The *loop* or *iteration* structure can be described as shown in the pseudocode in [Image 3](#).

Image 3. The loop structure in pseudocode.

Figure

```
Enter
  Test a condition for true or false
  Exit on false
  On true
    Perform one or more actions in sequence.
    Go back and test the condition again
```

Image 3. The loop structure in pseudocode.

As before, there is only one entry point and one exit point. Note that in this case, the exit point is not at the end of the pseudocode. Instead, it follows the test.

Perform the test and exit on false

The first thing that happens following entry is that a condition is tested for true or false.

If the test returns false, control simply exits the structure without taking any action at all.

Perform some actions and repeat the test on true

If the test returns true:

- One or more actions are performed in sequence.
- The condition is tested again and the process is repeated.

During each iteration, if the test returns false, control exits the structure. If the test returns true, the entire process is repeated.

Each action element may be another structure

Each of the action elements may be implemented by another sequence, selection, or loop structure.

Eventually all of the actions will be completed and the condition will be tested again.

Need to avoid infinite loops

Generally speaking, unless something is done in one of the actions to cause the test to eventually return false, control will never exit the loop.

In this case, the program will be caught in what is commonly referred to as an *infinite loop*.

Other possible structures

In some programming languages, there are structures other than sequence, selection, and loop that structured-programming experts are willing to accept for convenience including:

- The switch-case structure.
- The do-until structure.
- The for loop
- The for-each loop

While sometimes more convenient than the three main structures, these structures are not required for the solution of programming logic problems.

Preview

In this module, I will present and explain the simplest example of a selection structure that I was able to write in Scratch without using variables and without using relational or logical operators. *(I will explain operators, including relational and logical operators in future modules.)*

Screen output for the program named IfSimple01

The program places a basketball and two beach balls on the Stage as shown in [Image 4](#).

Image 4. User interface for the program named IfSimple01.

Figure



Image 4. User interface for the program named IfSimple01.

Click the green flag or the basketball

When the user clicks the green flag in the upper right corner, the three balls are placed in a horizontal line with the basketball in the center.

Scratch code, which can be paraphrased as shown in [Image 5](#), is executed each time the user clicks the basketball with the mouse.

Image 5. A paraphrased version of the Scratch code.

Figure

```
when Basketball is clicked{
  move basketball forward by 90 steps
  if(Basketball is touching RightBeachball){
    turn Basketball by 180 degrees
```

```
}//end if

if(Basketball is touching LeftBeachball){
    turn Basketball by 180 degrees
}//end if
}
```

Image 5. A paraphrased version of the Scratch code.

In other words, if you repetitively click the basketball with the mouse, it will move back and forth from left to right bouncing off of the two beach balls. The basketball will keep bouncing back and forth between the two beach balls for as long as you continue clicking on the basketball.

Discussion and sample code

Let's walk through the steps required to develop this program. I will deal first with the code that defines the behavior of the program when the user clicks the green flag in the upper right corner of [Image 4](#).

The Cartesian coordinate system The position of sprites on the Stage in the Scratch user interface is based on a two-dimensional [Cartesian coordinate system](#) with the origin at the center of the Stage.

Three sprites

Note in the lower right corner of [Image 4](#) that this program contains two beach balls and one basketball in addition to the Stage.

You add sprites to your program by clicking on the button with the icon of the file folder in the gray area immediately below the white Stage area in [Image 4](#).

When you click on one of the sprites that you have added to the program, that sprite appears at the top of the center pane. Having done that, you can then drag programming blocks into the center pane that define the behavior of that sprite.

Program code for the LeftBeachball

The code that I wrote (*by dragging blocks into the center pane*) for the sprite named **LeftBeachball** is shown in [Image 6](#).

Image 6. Program code for the left beach ball.

Figure

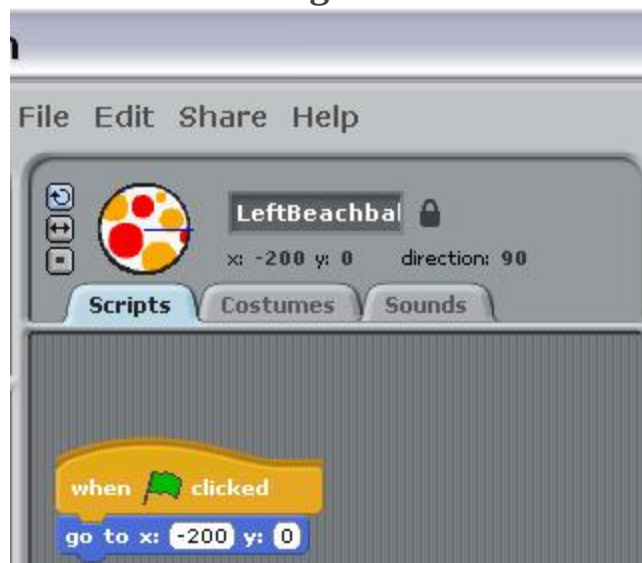


Image 6. Program code for the left beach ball.

You are already familiar with the tan block with the green flag shown in [Image 6](#) because you learned about it in a previous module. However, the

blue block in [Image 6](#) has not been used prior to this module.

Adding the blue *go to* block to the program

The blue block shown in [Image 6](#) was added to the program module by:

- Clicking the dark blue button labeled **Motion** in the upper left of [Image 4](#). (See [Image 13](#) for all of the programming blocks belonging to the Motion group.)
- Dragging the blue block shown in [Image 6](#) from the left panel to the center panel and clicking it into place under the tan block.
- Typing the literal values -200 and 0 into the two white boxes on the blue block.

Coordinate values

If you move the mouse pointer around in the user interface, the coordinates of the mouse pointer are displayed at the right end of the gray area immediately below the Stage as shown in [Image 4](#). The x coordinates range from -240 at the left to +240 at the right. The y coordinates range from +180 at the top to -180 at the bottom.

Behavior of the LeftBeachball

The behavior of the program module shown in [Image 6](#) can be interpreted as follows: When the user clicks the green flag, cause the sprite named **LeftBeachball** to move to a location with an x (*horizontal*) coordinate value of -200 and a y (*vertical*) coordinate value of 0. Since the origin is at the center of the Stage, this causes the beach ball to move to the left of the origin on the horizontal axis.

Some experimentation was required

Because I didn't know the diameter of the beach ball, I had to experiment to determine how far to move it to the left of the origin to locate it at the left side of the Stage as shown in [Image 4](#). I settled on a value of -200 for the x coordinate and a value of 0 for the y coordinate.

Positioning the other beach ball

In the interest of brevity, I won't show the code required to position the beach ball on the right side of the Stage. I did exactly the same thing for that beach ball except that I specified the value of the x coordinate to be 200 instead of - 200. This causes the beach ball named **RightBeachball** to move to the right side of the Stage when the user clicks the green flag.

The basketball

Initializing the position of the basketball

[Image 7](#) shows a portion of the center pane after clicking on the sprite named **Basketball**.

Image 7. Initializing the position and orientation of the basketball.

Figure



Image 7. Initializing the position and orientation of the basketball.

You will note that the tan block and the uppermost blue block in [Image 7](#) are the same as in [Image 6](#) except that the x coordinate value is set to 0. This causes the basketball to move to the origin when the user clicks the green flag.

Initializing the orientation of the basketball

However, [Image 7](#) contains a block that is not contained in [Image 6](#). The bottom blue block in [Image 7](#) is used to set the *orientation* of the basketball. *(By orientation, I mean the direction that the basketball is facing.)*

It may seem strange to say that a round basketball is facing in one direction or the other. However, every sprite has a front, back, top, and bottom even in those cases where it is not visually obvious. *(The orientation would be visually obvious if I were to use an animal for the sprite in place of the basketball. You can tell which direction the basketball is facing by observing the curved diagonal lines on the basketball.)*

After you drag the blue block labeled **point in direction** into the center pane, you can click the arrow in the white box to expose the following four choices:

- (90) right
- (-90) left
- (0) up
- (180) down

As you can see, I selected the choice that causes the basketball to face to the right. As a result, when the user clicks the green flag, the basketball will move to the origin and turn to face the right.

Handling mouse events on the basketball

[Image 8](#) shows all of the code in the center pane that applies to the basketball.

Image 8. All of the code that applies to the basketball.

Figure



Image 8. All of the code that applies to the basketball.

The top script in [Image 8](#) is another image of the script that was shown in [Image 7](#). At this point, we are interested in the behavior of the bottom script in [Image 8](#).

Behavior of the bottom script

The tan block labeled **when Basketball clicked** specifies that all of the actions produced by the blocks below that block will occur when the user clicks the basketball with the mouse. Furthermore, those actions will occur in top to bottom order.

The blue block labeled **move 90 steps** causes the basketball to move 90 steps forward (*in the direction that it is facing*). Each step constitutes one unit in the Cartesian coordinate system. For example, if you were to change this value from 90 to 240, that would cause the basketball to move from the origin to the extreme right edge of the Stage.

The selection block

As I indicated [earlier](#), the *selection* structure is sometimes referred to as a *decision* structure. In other words, this structure causes the program to select or make a decision between two alternatives. The pseudocode in [Image 9](#) describes this process.

Image 9. Pseudocode for a selection structure.

Figure

```
if a specified condition is true
  perform a specified action
otherwise (when the specified condition is not
true)
  perform a different action
```

Image 9. Pseudocode for a selection structure.

Sometimes the second part of the selection process isn't needed. In other words, in some cases, if the specified condition is not true, there is no requirement to do anything at all. That is the case in this program.

Two independent decisions

This program makes two independent decisions shown by the pseudocode in [Image 10](#).

Image 10. Two independent decisions.

Figure

```
if the basketball is touching the beach ball on  
the right
```

```
    turn around and face to the left
```

```
if the basketball is touching the beach ball on  
the left
```

```
    turn around and face to the right
```

Image 10. Two independent decisions.

These two decisions are implemented by the two tan blocks in [Image 8](#) containing the word **if** .

The Control group Clicking the tan **Control** button shown at the upper left in [Image 4](#) exposes a large number of programming blocks. Two of them are used to create pure selection structures and one is used to create a combination of a loop structure and a selection structure.

Selection control structures available in Scratch

Clicking the tan **Control** button shown at the upper left in [Image 4](#) exposes the blocks shown in [Image 11](#). The bottom two blocks can be used to create pure selection structures. The top block can be used to create a combination of a loop structure and a selection structure.

Selection structures are created by dragging these blocks into the center pane in order to apply them to a particular sprite.

Image 11. Selection control structures available in Scratch.

Figure

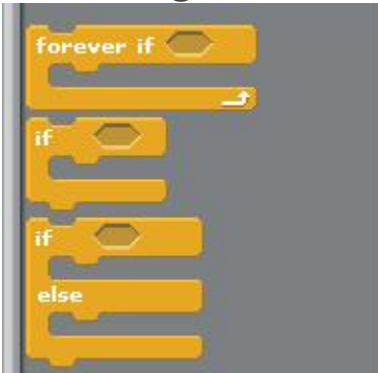


Image 11.
Selection control
structures
available in
Scratch.

How do the two bottom blocks differ?

The bottom block in [Image 11](#) is used to select between two specific actions. The middle block is used when there is only one action and you need to decide whether to take that action or not. That is the case in this program. If the basketball is touching a beach ball, a specific action is required. If the basketball is not touching a beach ball, no specific action is required.

Specifying the condition on which the decision will be based

Note the empty darker depressed area in each of the blocks shown in [Image 11](#) (the rectangle with the pointed ends) . In order to use either of these

blocks, you must drop another block having the same shape into the depressed area. The block that you drop into that area must specify the condition that will be used to make the decision.

Two groups of programming blocks have the correct shape

Unless I missed seeing some others, there are only two buttons at the top left of [Image 4](#) that expose blocks having the required shape:

- Sensing (light blue)
- Operators (green)

In this case, we will select and use a block from the light blue **Sensing** group. *(We will use programming blocks from the green **Operators** group in a future module.)* All of the blocks belonging to the **Sensing** group are shown in [Image 12](#).

Image 12. Programming blocks belonging to the Sensing group.

Figure

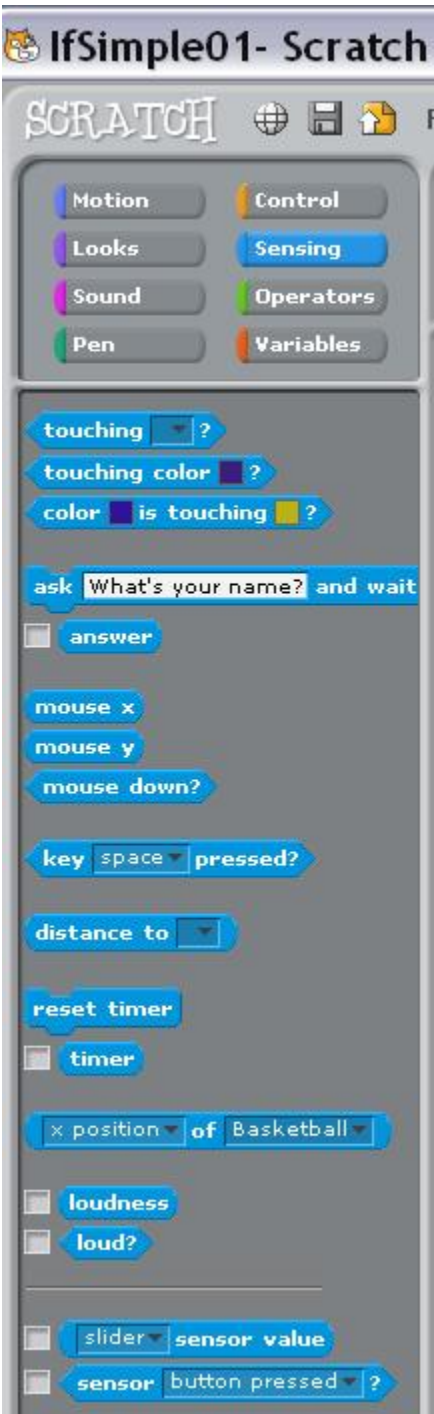


Image 12.
Programming blocks
belonging to the
Sensing group.

Will use the *touching* block

We will use the block labeled **touching** followed by a pull-down list and a question mark.

The items that appear in the pull-down list depend on the sprites that have been added to the program. For this program, that list consists of the following choices when the basketball has been selected for programming:

- mouse-pointer
- edge
- LeftBeachball
- RightBeachball

The top two choices are always there. The remaining choices depend on the sprites that have been added to the program at the point in time when you pull down the list and the sprite that has been selected for programming. *(Note that the **Basketball** sprite does not appear in the above list because it was selected for programming when I examined the list. In other words, you can't determine that the basketball is touching the basketball.)*

Go back and examine the script

Getting back to the bottom script in [Image 8](#), you can see that I dragged two copies of the **if** block shown in [Image 11](#) into the center panel and connected the blocks as shown in [Image 8](#). Then I dragged two copies of the **touching** block from the **Sensing** group shown in [Image 12](#), and dropped each of those blocks into the corresponding locations in the **if** blocks in [Image 8](#).

Then I selected **RightBeachball** from the pull-down list for one of the **touching** blocks and selected **LeftBeachball** from the pull-down list for the other **touching** block.

Conditions have been established - need actions

At this point, I had the conditions for the two selection structures established, but I hadn't yet specified the actions to be taken when one or the other of the conditions is found to be true.

Programming blocks belonging to the Motion group

[Image 13](#) shows the programming blocks that are exposed by clicking the **Motion** button at the top left of [Image 4](#).

Image 13. Programming blocks belonging to the Motion group.

Figure

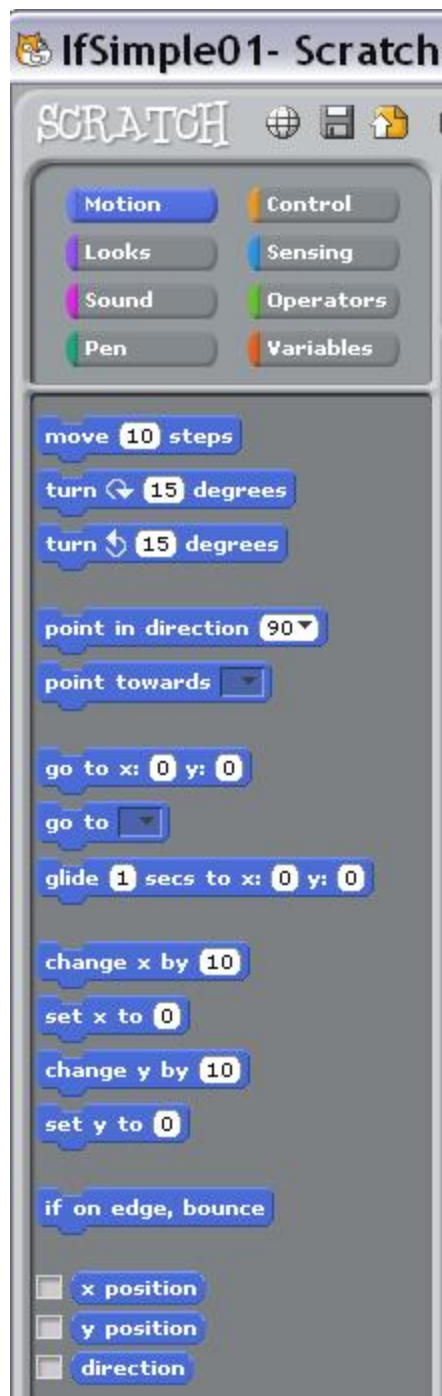


Image 13.
Programming blocks
belonging to the
Motion group.

You saw three of the blocks from [Image 13](#) being used in [Image 6](#), [Image 7](#), and immediately below the top tan block in the bottom script in [Image 8](#). Now we need to use another of the blocks from [Image 13](#).

Turn around and face the other way

Recall that the blue **point in direction** block in the top script in [Image 8](#) causes the basketball to turn to face to the right when the user clicks the green flag.

The two **turn** blocks in the bottom script in [Image 8](#) cause the basketball to rotate around its center by 180 degrees. This, in turn, causes it to face in the opposite direction from the direction that it was previously facing. *(It also turns it upside down, but that doesn't matter for a round basketball.)*

This is the action that is required whenever either of the **touching** blocks is true. In other words, whenever the basketball touches either of the beach balls, it must turn to face the opposite direction and be prepared to move 90 steps in that direction the next time the user clicks the basketball.

An online version of this program is available

A copy of this program has been posted online for your review (see [Resources](#) for the URL) . If you don't find the program using that URL, search the Scratch site for the user named dbal.

Run the program

I encourage you to use the information provided above to write this program. Experiment with the code, making changes, and observing the results of your changes. Make certain that you can explain why your changes behave as they do.

Just for fun, use blocks from the purple **Sound** group and add some sound effects to your program.

I also encourage you to write the program described below.

Student programming project

Write a Scratch program named **IfWithVar01** that produces the output shown in [Image 14](#) when the user clicks the green flag.

Image 14. Initial output from the program named IfWithVar01.

Figure

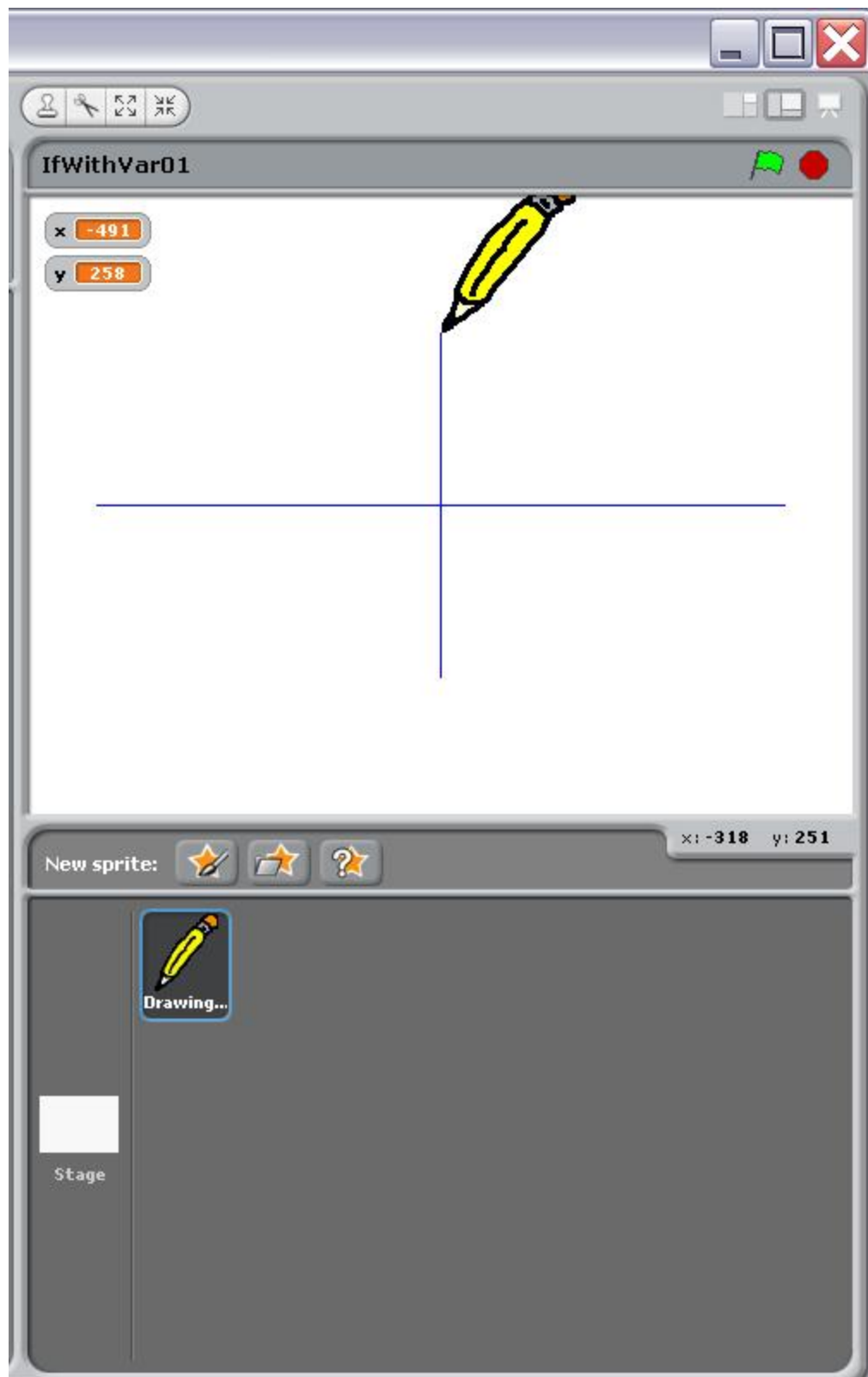


Image 14. Initial output from the program named

IfWithVar01.

Orthogonal axes in Cartesian coordinates

When the user clicks the green flag, a **DrawingPencil** sprite draws a pair of orthogonal axes that intersect at the origin in the white Stage area. Make the horizontal axis extend from -200 to 200. Make the vertical axis extend from -100 to 100.

Draw a straight line to the location of the next mouse click

Each time the user clicks the mouse in the white Stage area (*after the user has clicked the green flag*), a straight line is drawn from the current location of the **DrawingPencil** to the location where the mouse click occurred. [Image 15](#) shows an example output after two mouse clicks. Image 15. Program output after having clicked twice in the Stage area.

Figure

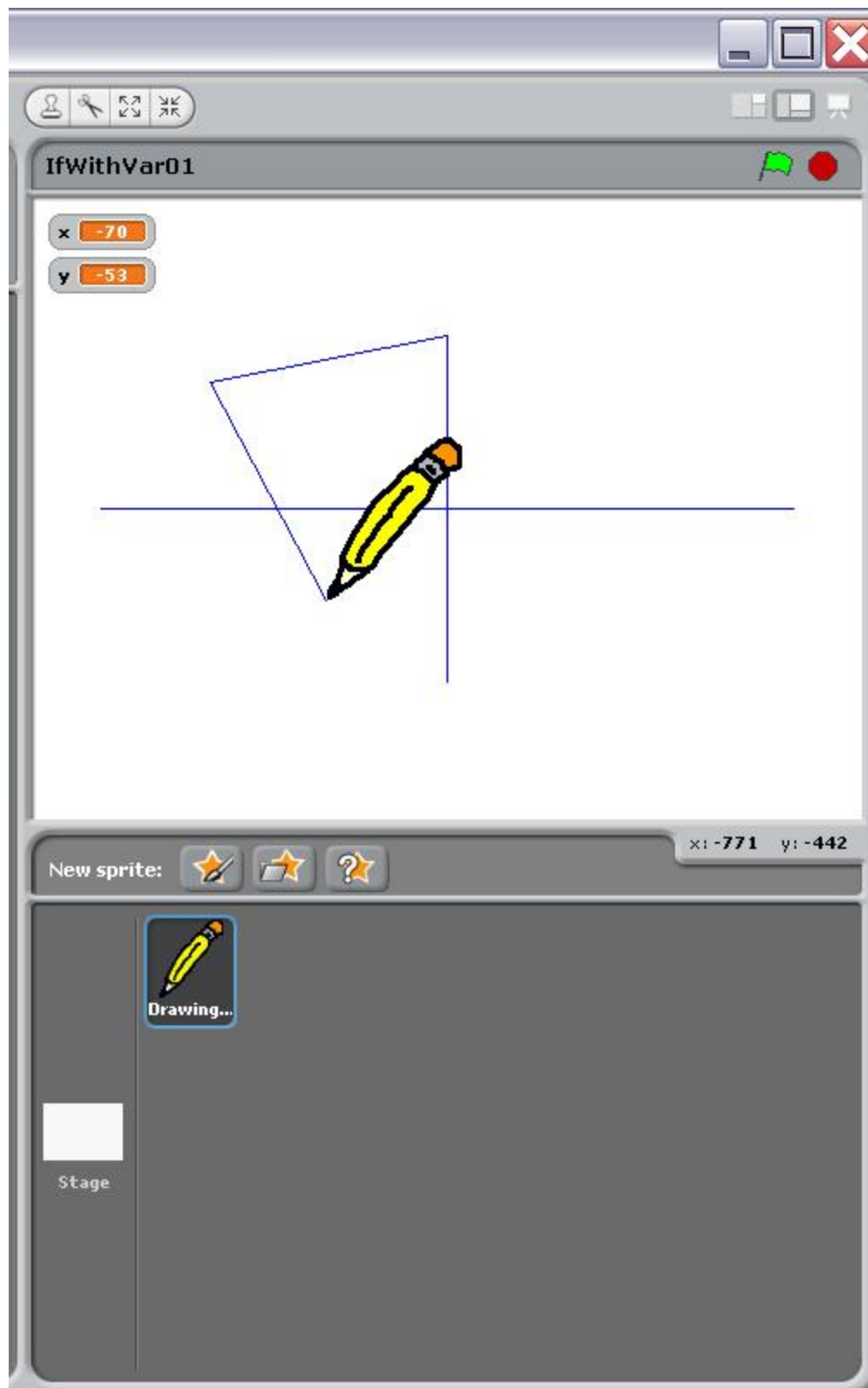


Image 15. Program output after having clicked twice in

the Stage area.

A sneak peek at the solution

In case you need to sneak a peek at the solution to this programming project, a copy of this program has been posted online for your review (see [Resources](#) for the URL) . (If you don't find the program using that URL, search the Scratch website for the user named **dbal** .)

Once you locate the project on the Scratch web site, you can execute it online. You can also download the project and open it in the Scratch Programming Environment on your computer by following the instructions under **Download this project!**

Summary

I began by explaining structured programming, the sequence structure, the selection structure, and the loop structure. Then I presented and explained a Scratch program that illustrates the selection structure. (*The sequence structure is so simple that it doesn't require an explanation. The loop structure will be explained in a future module.*) The program also illustrates the handling of mouse events and Cartesian coordinates.

Finally, I provided the specifications for a student-programming project for you to write to demonstrate your understanding of what you learned from the first program.

Copies of both programs have been posted online for your review (see [Resources](#) for the URL) . If you don't find the program using that URL, search the Scratch site for the user named **dbal** .

What's next?

In the next module, I will teach you about arithmetic operators. In the two modules following that one, I will teach you about relational and logical operators and how to use those operators to write the conditional expressions used in selection and loop structures.

Resources

- [Scratch home](#)
- [Scratch download page](#)
- [Scratch tutorial - Dance Tutorial](#)
- [Scratch forums](#)
- [Son of String Art](#)
- [Scratch explanatory video](#)
- [Scratch tutorials from MIT](#)
- [Scratch tutorial - Digital Art for All](#)
- [Scratch for Budding Computer Scientists](#)
- [Learn Scratch at learnscratch.org](#)
- [Scratch Tutorial - Space Shuttle Mission STS-2020](#)
- [SCRATCH TUTORIALS ICT In Primary Education 2012-2013](#)
- [Scratch Wiki](#)
- [Scratch WikiTable of ContentsWebsite - Scratch Wiki](#)
- [Scratch WikiTable of ContentsTutorials - Scratch Wiki](#)
- [Scratch WikiTable of ContentsProgram - Scratch Wiki](#)
- [Tutorial about Variables - Scratch Wiki](#)
- [Scratch support MIT web site](#)
- [Scratch resources at Scratch.ie](#)
- [ScratchEd Resources](#)
- [Scratch project Day Dream](#)
- [Scratch project Son of String Art](#)
- [Scratch Project Scratch Tutorial](#)
- [Variable01](#) - Online version of program
- [Variable02](#) - Online version of student programming project
- [Variable03](#) - Online version of student programming project
- [IfSimple01](#) - Online version of program
- [IfWithVar01](#) - Online version of student programming project

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Scr0130: Sequence, Selection, and Loop
- File: Scr0130.htm
- Published: 03/26/13

Note: Disclaimers:

Financial : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.

I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module. In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.

Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Scr0140: Arithmetic Operators

The purpose of this module is to teach you about operators and operands in general and arithmetic operators in particular. You will also learn about expressions and statements, and you will learn how to write a Scratch program that illustrates the use of arithmetic operators in Scratch.

Table of Contents

- [Preface](#)
 - [Viewing tip](#)
 - [Images](#)
- [General background information](#)
 - [Operators](#)
 - [Operands](#)
 - [Expressions](#)
 - [Statements](#)
 - [A brief word about type](#)
 - [Unary, binary, and ternary operators](#)
 - [Some operators can be either unary or binary.](#)
 - [The minus character as a unary operator](#)
 - [Binary operators use infix notation](#)
 - [General behavior of an operator](#)
 - [Operator categories](#)
- [Preview](#)
- [Discussion and sample code](#)
 - [Four variables](#)
 - [A button](#)
 - [Two scripts in the center panel](#)
 - [Variables with sliders](#)

- [How to create a slider](#)
- [Full-size view of the center panel](#)
- [Initialize the variable values to zero](#)
- [Define the behavior of the button](#)
 - [Constructing the script](#)
 - [Not exactly what we want to happen](#)
- [The Operators panel](#)
 - [Four arithmetic operators](#)
 - [Use an asterisk for multiplication](#)
 - [Use a forward slash for division](#)
 - [More blocks of the correct shape](#)
 - [Random numbers](#)
 - [The modulus](#)
 - [Various numeric representations](#)
 - [Rounding a number](#)
- [Drag and drop the plus and minus operators](#)
- [A few more steps are required](#)
- [Operation of the program](#)
- [An online version of this program is available](#)
- [Run the program](#)
- [Student programming project](#)
- [Summary](#)
- [What's next?](#)
- [Resources](#)
- [Miscellaneous](#)

Preface

This module is one in a collection of modules designed to help beginners of all ages (*8 and up*) learn how to create the code for computer programs. Information is provided not only for the beginners themselves but also for their parents and teachers where appropriate.

The purpose of this module is to teach you about operators and operands in general and arithmetic operators in particular. You will also learn about expressions and statements, and you will learn how to write a [Scratch](#) program that illustrates the use of arithmetic operators in Scratch.

I will also provide the specifications for a student-programming project for you to complete in order to demonstrate your understanding of what you learned from the first program.

Viewing tip

I recommend that you open another copy of this module in a separate browser window and use the following links to easily find and view the images while you are reading about them.

Images

- [Image 1](#). Reduced screen shot of Arithmetic01 in operation.
- [Image 2](#). Full-size view of the center panel.
- [Image 3](#). Preparing to use the arithmetic operators.
- [Image 4](#). Result of dropping addition and subtraction operators into variable blocks.
- [Image 5](#). Stage area of the finished program.
- [Image 6](#). Making the button say ouch.
- [Image 7](#). Output from student project program named Arithmetic02.

General background information

Operators

Operators are the action elements of a computer program. They perform actions such as adding two variables, dividing one variable by another variable, comparing one variable to another variable, etc.

Operands

According to the current jargon, *operators* operate on *operands* .

For example, in the following expression, the plus character is an operator while **x** and **y** are operands.

x + **y**

Assuming that **x** and **y** are numeric variables, this expression produces the sum of the values stored in the variables named **x** and **y** . The variable **x** would be called the *left operand* and the variable **y** would be called the *right operand* .

Expressions

Computer programs in many languages consist of statements, which in turn, consist of expressions. (*Programming blocks substitute for statements in Scratch.*)

Expressions An expression is a specific combination of operators and operands that evaluates to a particular result. The operands can be variables, literals, or method calls. (*Note that Scratch doesn't support methods.*)

In your past experience, you may have referred to expressions by the names *formulas* or *equations* . Although formulas and equations are not exactly the same thing as expressions, they are close enough to help you understand what expressions are and how they are used.

Statements

A statement is a specific combination of expressions. The following is an example of a statement comprised of expressions in a text-based language such as Java, C++, or C#.

```
z = x + y;
```

Operationally, values are retrieved from the variables named **x** and **y** in the above statement. These two values are added together. The result is stored in (*assigned to*) the variable named **z** , replacing whatever value may previously have been contained in that variable.

The plus character (+) would commonly be referred to as the addition operator or the concatenation operator, depending on the *type* of data stored in the variables named **x** and **y** . The equal character (=) would commonly be called the assignment operator in Java, Alice, C#, and C++, but we will see in a future module that it is also used as a *relational* operator in Scratch.

A brief word about type

As a very crude analogy, you can think of *variables* as the pens at the animal shelter and think of *type* as the kinds of animals that reside in those pens. Dogs and cats are different types of animals that usually don't coexist very well in the same pen. Therefore, they are normally put in different pens.

Similarly, different types of data don't coexist well in the same variable in *type-sensitive* languages. Therefore, in type-sensitive languages, there are very stringent rules as to the types of data that can be stored in each variable.

Scratch has two types of data that can be stored in a variable (*numeric and string*) . However, Scratch is not a *type-sensitive* language so type is not an issue in Scratch. (*Scratch also has a semblance of a boolean type but it tends to take care of itself.*)

Unary, binary, and ternary operators

Many programming languages provide operators that can be used to perform an action on one, two, or three operands. I believe that operators in Scratch are confined to only one or two operands.

An operator that operates on one operand is called a **unary** operator. An operator that operates on two operands is called a **binary** operator. An operator that operates on three operands is called a **ternary** operator.

Some operators can be either unary or binary

Some operators can behave either as a unary or as a binary operator in languages such as Java and C#. However, I don't believe that this is the case in Scratch. The best-known operator that can behave either way in those other languages is the minus character (-).

As a binary operator, the minus character causes its right operand to be subtracted from its left operand. For example, the third statement below subtracts the variable **y** from the variable **x** and assigns the result of the subtraction to the variable **z** . After the third statement is executed, the variable **z** contains the value 1.

```
int x = 6;
```

```
int y = 5;
```

```
int z = x - y;
```

The minus character as a unary operator

As a unary operator, the minus character causes the algebraic sign of the right operand to be changed. For example, the second statement below causes a value of -5 to be assigned to (*stored in*) the variable **x** .

```
int y = 5;
```

```
int x = -y;
```

Binary operators use infix notation

To keep you abreast of the current jargon, **binary** operators in Scratch always use *infix* notation. This means that the operator appears between its operands.

Some other programming languages have **unary** operators that use *prefix* notation and *postfix* notation. For prefix notation, the operator appears before (*to the left of*) its operand. For postfix notation, the operator appears after (*to the right of*) its operand.

General behavior of an operator

As a result of performing the specified action, an operator can be said to return a value (*or evaluate to a value*) of a given type. The type of value returned depends on the operator and the type of the operands.

To evaluate to a value: To evaluate to a value means that after the action is performed, the operator and its operands are effectively replaced in the expression by the value that is returned.

Operator categories

There are many different kinds of operators. Therefore, the easiest way to study them is to divide them into categories such as the following

- arithmetic
- relational
- logical

- bitwise
- assignment

This module will concentrate on arithmetic operators. Future modules will deal with other kinds of operators.

Preview

In this module, I will present and explain a Scratch program named **Arithmetic01** . This program illustrates the use of the following arithmetic operators (see [Image 4](#)):

- + (addition)
- - (subtraction)

Variables with the following names are created and displayed on the screen (see [Image 5](#)):

- **LeftOperand** - has a slider
- **RightOperand** - has a slider
- **Sum**
- **Diff**

In addition, a button is displayed on the screen.

The user adjusts the values of the **LeftOperand** and **RightOperand** variables with a pair of sliders. When the user clicks the button, the variable named **Sum** displays the sum of the values of the left and right operands. The variable named **Diff** displays the result of subtracting the right operand from the left operand.

Discussion and sample code

I'm going to walk you through the steps required to develop this program, being brief on those things that you already know about and being more verbose on the new material.

To begin with, [Image 1](#) shows a reduced screen shot of the user interface. Image 1. Reduced screen shot of Arithmetic01 in operation.

Figure

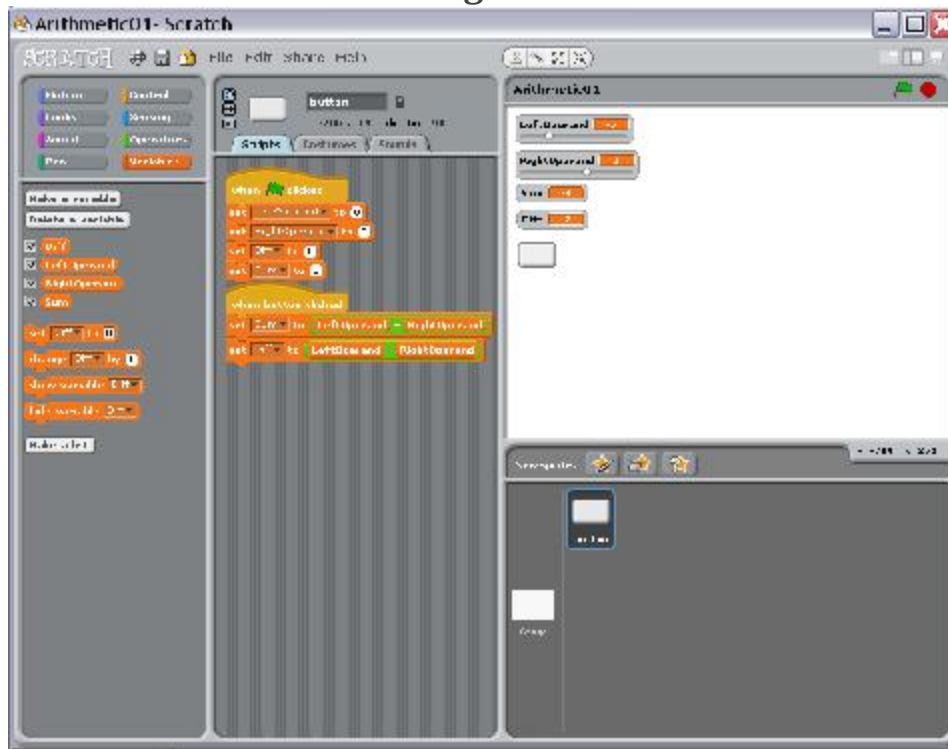


Image 1. Reduced screen shot of Arithmetic01 in operation.

I realize that [Image 1](#) lacks detail due to the requirement to reduce the size of the screen shot for publication in this format, but there are some aspects of [Image 1](#) that are worth noting. I will go from left to right in [Image 1](#) making comments.

Four variables

The leftmost panel shows that four variables have been created. Because you can't read the names of those variables, I will list them here:

- **LeftOperand**
- **RightOperand**
- **Sum**
- **Diff**

You can barely see that the checkboxes next to all four variables have been checked. As you learned in an earlier module, checking this box causes a variable to be displayed in the white Stage area on the right. Thus you can see all four variables being displayed in the upper left corner of the Stage.

A button

Skipping to the area immediately below the Stage, you can (*almost*) see that a button has been added to the program. That button has been selected in [Image 1](#), making it possible to drag blocks into the center panel that controls the behavior of the button.

Two scripts in the center panel

Although you can't read the details in [Image 1](#), you can see that there are two scripts showing in the center panel. (*I will show you a full-size screen shot of the center panel later in [Image 2](#).*)

For now, suffice it to say that the top script initializes the values stored in all four variables when the user clicks the green flag in the upper right of [Image 1](#). The bottom script in the center panel defines the behavior of the program when the user clicks the button in the Stage area.

Variables with sliders

If you look carefully, you can tell that the top two variables that are displayed in the Stage area look different from the bottom two variables. This is because a slider has been assigned to each of the top two variables

(see [Image 5](#)) . This makes it possible for the user to manually set the values stored in each of these two variables.

How to create a slider

To create a slider for a variable, right click on the display of the variable in the Stage area and select **slider** in the popup menu that appears.

Once you have caused a slider to appear with the variable display, you can right-click on the variable display again and select **set slider min and max** in the popup menu to specify the range of the slider. This will cause a simple dialog box to appear into which you can enter the minimum value and the maximum value and then click an OK button. In this program, I have both sliders set to a minimum value of -10 and a maximum value of +10.

Full-size view of the center panel

[Image 2](#) shows a full-size view of the center panel after having selected the button icon in the area immediately below the Stage in [Image 1](#).

Image 2. Full-size view of the center panel.

Figure



Image 2. Full-size view of the center panel.

Initialize the variable values to zero

As I mentioned earlier, the top script in [Image 2](#) sets the value of each of the four variables to zero when the user clicks the green flag. The code in that script should be completely familiar to you by now and no explanation should be necessary.

Define the behavior of the button

The bottom script in [Image 2](#) defines the behavior of the program when the button is clicked. Basically, that code says to set the value of the variable named **Sum** to the sum of the contents of the two variables named **LeftOperand** and **RightOperand** and to set the value of the variable named **Diff** to the value of the left operand minus the value of the right operand.

Constructing the script

That seems straightforward enough. However, the process of constructing that script contains some new material, so I will walk you through the process.

[Image 3](#) shows the state of the development process immediately after the blocks for the variables named **Sum** and **Diff** have been dragged from the **Variables** panel to the center panel and connected to the block that reads **when button clicked**.

Image 3. Preparing to use the arithmetic operators.

Figure

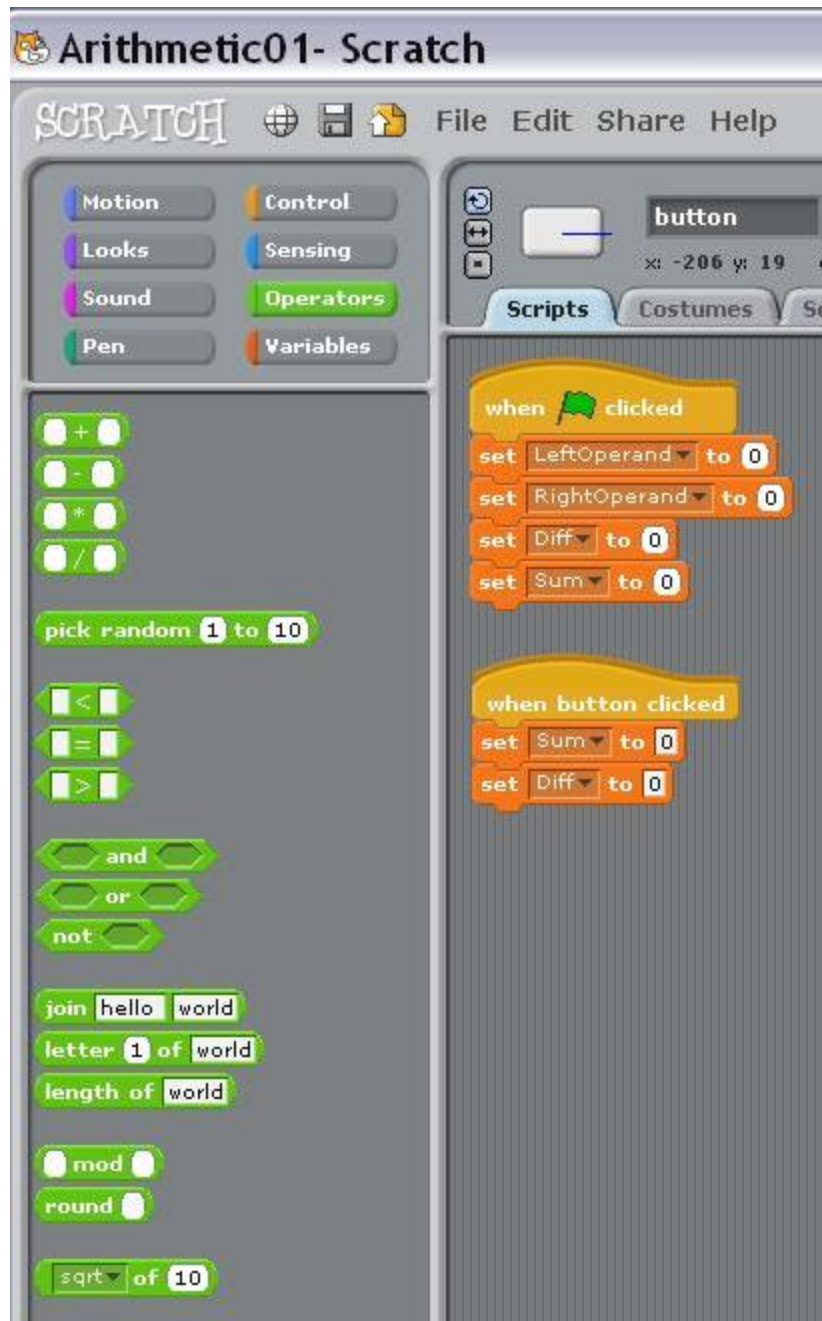


Image 3. Preparing to use the arithmetic operators.

Not exactly what we want to happen

In [Image 3](#), as it stands now, the variables named **Sum** and **Diff** will be set to a value of zero when the user clicks the button. That isn't what we want, so we have some more programming to do.

The desired behavior is for the value of the variable named **Sum** to be set to the sum of the variables named **LeftOperand** and **RightOperand** when the user clicks the button. Similarly, we want the value of the variable named **Diff** to be set to the difference between those two variables.

The *Operators* panel

The left panel in [Image 3](#) shows the blocks that are exposed when the green button labeled **Operators** is clicked.

Eleven of those blocks are of the correct shape to fit into the white boxes in the two bottom orange blocks in the center panel. This means that we could drag any one of those eleven blocks and drop them into those white boxes.

Four arithmetic operators

The top four green blocks in the left panel of [Image 3](#) are particularly interesting. They contain the following arithmetic symbols as part of their labels:

- +
- -
- *
- /

These are the symbols that are commonly used for addition, subtraction, multiplication, and division in computer programming. The top two match what you are accustomed to seeing on your hand calculator, but the bottom two probably don't match.

Use an asterisk for multiplication

Most programming languages use the `*` character instead of the **X** character for multiplication in order to keep the multiplication operator from being confused with the letter that uses the **X** symbol.

Use a forward slash for division

Insofar as division is concerned, most computer keyboards don't have a symbol that matches the symbol commonly used for division on hand calculators, so the `/` character is used instead.

More blocks of the correct shape

Before leaving the discussion of the left panel in [Image 3](#), I will explain the behavior of four additional blocks that are of the correct shape for being dropped into the white boxes in the variable blocks in the center panel. *(The remaining blocks beyond those four deal with strings, which I will ignore for now.)*

Random numbers

The block labeled **pick random 1 to 10** lets you enter other values in place of 1 and 10. Then this block will deliver a random number within that range when it is called upon to do so. Random numbers are often used in game programs to simulate the throwing of dice or the spinning of a wheel of fortune.

The modulus

The block labeled **mod** will produce the modulus of two values when called upon to do so. The modulus of two numbers is the remainder that results from dividing one number by another number. You may remember the

remainder from when you learned to do long division in elementary school but before you learned about decimals.

Various numeric representations

The block labeled **sqrt** will deliver about a dozen different representations of numeric values, depending on what is selected from the pull-down list. This includes the square root, trigonometric functions, absolute value, etc.

Rounding a number

Finally, the block labeled **round** will round a decimal number to the nearest whole number.

Drag and drop the plus and minus operators

[Image 4](#) shows the result of dragging the top two green blocks from the left panel and dropping them into the white boxes in the bottom two orange variable blocks in the center panel of [Image 3](#).

Image 4. Result of dropping addition and subtraction operators into variable blocks.

Figure

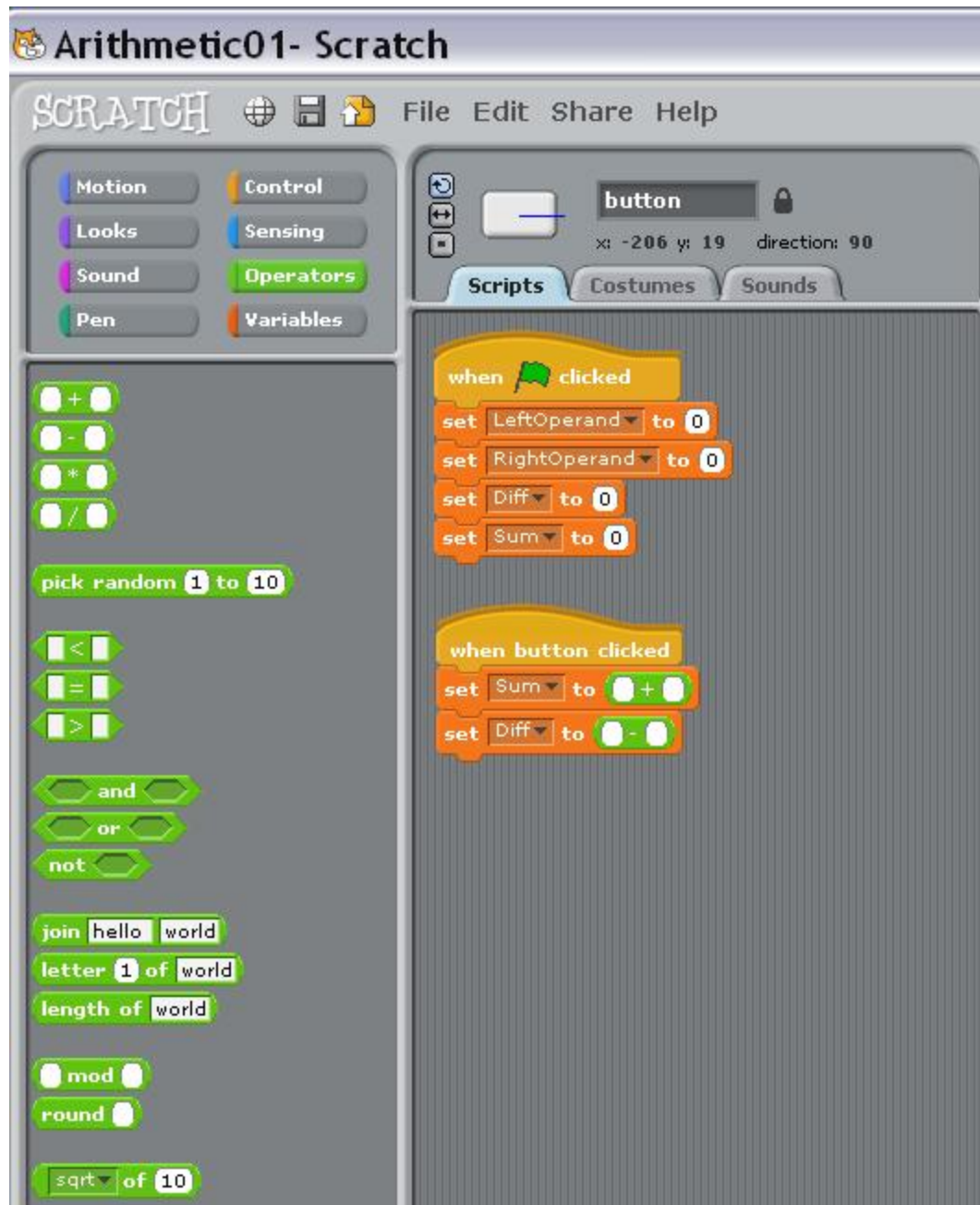


Image 4. Result of dropping addition and subtraction operators into variable blocks.

A few more steps are required

If you compare [Image 4](#) with [Image 2](#), you will see that [Image 4](#) is close to what we need but we aren't quite there yet. In order to cause the bottom two blocks in the center panel in [Image 4](#) to match the bottom two blocks in [Image 2](#), we need to do the following:

- Click the orange **Variables** button in the upper left of [Image 4](#) to expose the variables as shown in [Image 1](#).
- Drag the blocks for the variables named **LeftOperand** and **RightOperand** from the left panel and drop them into the white boxes in the bottom two blocks in [Image 4](#) to make them match the bottom two blocks in [Image 2](#).

That's the solution. The upper left portion of your Stage area should now look similar to [Image 5](#). Note however that you may need to use the mouse to arrange the four variables and the button to get your Stage arranged like [Image 5](#).

Image 5. Stage area of the finished program.

Figure



Image 5. Stage
area of the
finished
program.

Operation of the program

Once you reach this point, you can click the green flag to initialize all four variables to zero.

Then you can move the sliders back and forth to manually set the values for the variables named **LeftOperand** and **RightOperand** .

Then when you click the button at the bottom of [Image 5](#), the variable named **Sum** will display the sum of the values of the top two variables, and the variable named **Diff** will display the value of **LeftOperand** minus the value of **RightOperand** .

An online version of this program is available

A copy of this program has been posted online for your review (*see [Resources](#) for the URL*) . If you don't find the program using that URL, search the Scratch site for the user named dbal.

Run the program

I encourage you to use the information provided above to write this program. Experiment with the code, making changes, and observing the results of your changes. Make certain that you can explain why your changes behave as they do.

Just for fun, move the button a little to the right and cause it to say **Ouch** for about five seconds each time you click it as shown in [Image 6](#). *Hint: See the purple button labeled Looks in [Image 3](#).*

Image 6. Making the button say ouch.

Figure

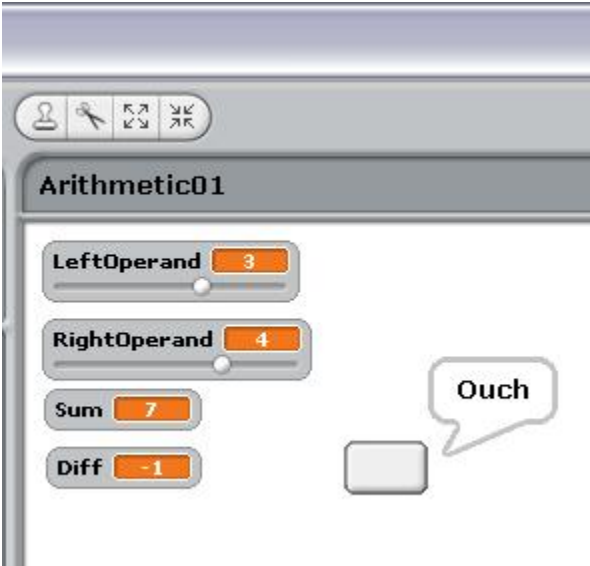


Image 6. Making the button say ouch.

I also encourage you to write the program described below.

Student programming project

Write a Scratch program named **Arithmetic02** that produces the output shown in [Image 7](#) when the user adjusts the sliders to the values shown and clicks the button. Make the word **Ouch!** appear and then go away after about five seconds.

Image 7. Output from student project program named Arithmetic02.

Figure



Image 7. Output from student project program named Arithmetic02.

A copy of this program has been posted online for your review (*see [Resources](#) for the URL*). If you don't find the program using that URL, search the Scratch site for the user named **dbal**.

Summary

I began by teaching you about operators and operands in general. I also taught you about expressions and statements. I gave a very brief introduction to type.

I presented and explained a sample Scratch program that illustrates how to use arithmetic operators in Scratch.

Finally, I provided the specifications for a student-programming project for you to write in order to demonstrate your understanding of what you learned from the first program.

Copies of both programs have been posted online for your review (*see [Resources](#) for the URL*) . If you don't find the program using that URL, search the Scratch site for the user named dbal.

What's next?

This module concentrated on arithmetic operators. The next few modules will deal with relational and logical operators as well as selection and loops.

Resources

- [Scratch home](#)
- [Scratch download page](#)
- [Scratch tutorial - Dance Tutorial](#)
- [Scratch forums](#)
- [Son of String Art](#)
- [Scratch explanatory video](#)
- [Scratch tutorials from MIT](#)
- [Scratch tutorial - Digital Art for All](#)
- [Scratch for Budding Computer Scientists](#)
- [Learn Scratch at learnscratch.org](#)
- [Scratch Tutorial - Space Shuttle Mission STS-2020](#)
- [SCRATCH TUTORIALS ICT In Primary Education 2012-2013](#)
- [Scratch Wiki](#)
- [Scratch WikiTable of ContentsWebsite - Scratch Wiki](#)
- [Scratch WikiTable of ContentsTutorials - Scratch Wiki](#)
- [Scratch WikiTable of ContentsProgram - Scratch Wiki](#)
- [Tutorial about Variables - Scratch Wiki](#)
- [Scratch support MIT web site](#)
- [Scratch resources at Scratch.ie](#)

- [ScratchEd Resources](#)
- [Scratch project Day Dream](#)
- [Scratch project Son of String Art](#)
- [Scratch Project Scratch Tutorial](#)
- [Variable01](#) - Online version of program
- [Variable02](#) - Online version of student programming project
- [Variable03](#) - Online version of student programming project
- [IfSimple01](#) - Online version of program
- [IfWithVar01](#) - Online version of student programming project
- [Arithmetic01](#) - Online version of program
- [Arithmetic02](#) - Online version of student programming project

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Scr0140: Arithmetic Operators
- File: Scr0140.htm
- Published: 03/28/13

Note: Disclaimers:

Financial : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.

I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module. In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com

showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.

Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Scr0150: Relational Operators

The purpose of this module is to teach you how to write a Scratch program that uses the following relational operators: less than, equal to, and greater than.

Table of Contents

- [Preface](#)
 - [Viewing tip](#)
 - [Images](#)
- [General background information](#)
 - [Operator categories](#)
- [Preview](#)
 - [Program operation](#)
 - [No boolean type available in Scratch](#)
- [Discussion and sample code](#)
 - [A button and five variables](#)
 - [The program code](#)
 - [Initialize the variables](#)
 - [When the button is clicked...](#)
 - [An if-else block](#)
 - [Need to put a conditional clause in the pocket](#)
 - [Where are the blocks with the correct shape?](#)
 - [Will use blocks from the Operators group](#)
 - [Six blocks have the correct shape](#)
 - [Relational operators in other languages](#)
 - [Three steps at once](#)
 - [Still need to complete the conditional clause](#)
 - [Complete the relational expression](#)

- [Completing the program](#)
 - [A screen shot of the program output](#)
 - [Interpretation of the results](#)
 - [An online version of this program is available](#)
- [Run the program](#)
- [Student programming project](#)
 - [Operation of the program](#)
- [Summary](#)
- [What's next?](#)
- [Resources](#)
 - [General resources](#)
 - [Programs used in this collection](#)
- [Miscellaneous](#)

Preface

This module is one in a collection of modules designed to help beginners of all ages (*8 and up*) learn how to create the code for computer programs. Information is provided not only for the beginners themselves but also for their parents and teachers where appropriate.

The purpose of this module is to teach you how to write a Scratch program that uses the following relational operators: ***less than*** , ***equal to*** , and ***greater than*** .

Viewing tip

I recommend that you open another copy of this module in a separate browser window and use the following links to easily find and view the images while you are reading about them.

Images

- [Image 1](#). Reduced screen shot of program Relational01 in operation.
- [Image 2](#). The center programming panel for Relational01.
- [Image 3](#). Starting to use an if-else block.
- [Image 4](#). Green blocks exposed by clicking the Operators button.
- [Image 5](#). Intermediate stage in construction of if-else script.
- [Image 6](#). Screen shot of the output from the program named Relational01.
- [Image 7](#). Screen shot of the output from the program named Relational02.

General background information

First, a quick review of material from earlier modules:

- **Operators** are the action elements of a computer program. They perform actions such as adding two variables.
- **Operands** are the things that are operated on by operators. For example, variables are often the operands that are operated on by operators.
- An **expression** is a specific combination of operators and operands, which evaluates to a particular result.
- A **statement** is a specific combination of expressions.
- The equal character (=) would commonly be called the **assignment** operator in programming languages such as Java but we will see later that it is used as a *relational* operator in Scratch.
- Scratch has two types of data (*numeric and string*).
- An operator that operates on one operand is called a **unary** operator.
- An operator that operates on two operands is called a **binary** operator.
- An operator that operates on three operands is called a **ternary** operator. Scratch doesn't have any ternary operators.
- **Binary** operators in Scratch use *infix* notation. This means that the operator appears between its operands.

Operator categories

There are several different kinds of operators. The easiest way to study them is to divide them into categories such as the following

- arithmetic
- relational
- logical
- bitwise
- assignment

An earlier module explained *arithmetic* operators. This module will explain *relational* operators. Future modules will deal with the other kinds of operators.

A previous module introduced you to the *selection* structure. This module will expand on that concept.

Preview

In this module, I will present and explain a Scratch program named **Relational01** . This program illustrates the use of the following relational operators:

Note:

< (less than)
= (equal to)
> (greater than)

The program creates the following five variables and a button and displays them on the Stage **as shown** in [Image 1](#) and [Image 6](#):

- LeftOperand - a slider

- RightOperand - a slider
- LessThan
- Equals
- GreaterThan

Image 1. Reduced screen shot of program Relational01 in operation.
Figure

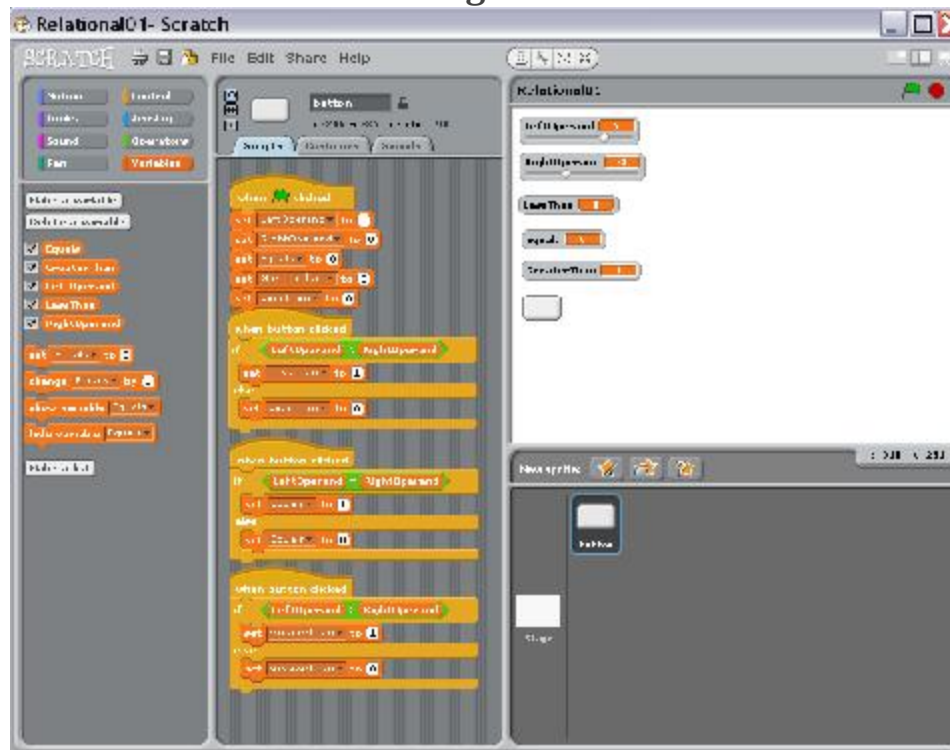


Image 1. Reduced screen shot of program Relational01 in operation.

Program operation

When the user clicks the green flag, the values of the variables shown in [Image 6](#) are set to 0.

The user slides the two sliders to set the values of **LeftOperand** and **RightOperand** .

When the user clicks the button, three separate event handlers on the button test the left operand against the right operand for *less than* , *equal to* , and *greater than* and display the results in the three variables having the corresponding names. A 0 is displayed if the value is false and a 1 is displayed if the value is true. Those three event handlers are shown as the bottom three scripts in the center panel of [Image 1](#) .

No boolean type available in Scratch

Many other programming languages have a **boolean** type. The *boolean* type can take on values of **true** and **false** . Scratch doesn't have a *boolean* type for variables. (*The only variable types in Scratch are numeric and string.*)

The boolean type is commonly used in some kind of a test to determine what to do next.

Because of the lack of a *boolean* type in Scratch, you must improvise. As you will see in this program, we will use a numeric value of 0 to represent **false** and a numeric value of 1 to represent **true** .

We could also have used the string "False" to represent **false** and the string "True" to represent **true** . However, **0** and **1** have historically been used to represent **false** and **true** in programming languages that didn't have a true boolean type so I decided to use that convention here also.

Discussion and sample code

A button and five variables

As you can see in the bottom right of [Image 1](#), a button was added to this program. If you look very carefully at the left panel in [Image 1](#), you will see that five variables were created for the program and that all five of the variables were displayed in the Stage area in the upper right. *(The checkboxes for all five variables were checked causing them to appear in the Stage area.)* A better view of the Stage is provided in [Image 6](#).

The names of the five variables were listed [above](#). By this point, you should have no difficulty creating variables and causing them to be displayed in the Stage, so no explanation should be necessary.

The program code

[Image 2](#) shows a full-size view of the center panel in [Image 1](#). This panel contains the program code associated with the button.

Image 2. The center programming panel for Relational01.

Figure

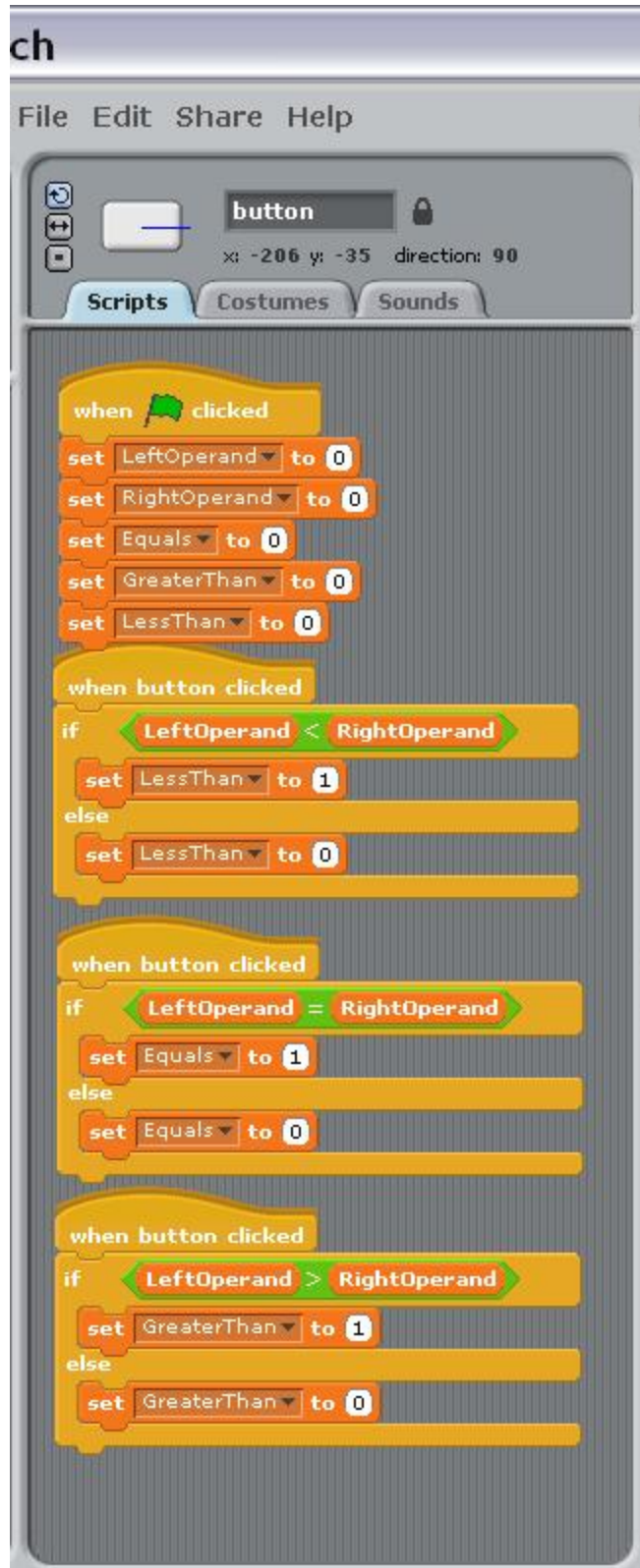


Image 2. The center programming

panel for Relational01.

Initialize the variables

The top script in [Image 2](#) initializes the values of all the variables to zero when the user clicks the green flag. You have seen code like this several times before in this series of modules, so I won't discuss it further.

When the button is clicked ...

Each of the bottom three scripts in [Image 2](#) are executed when the user clicks the button in the Stage area in [Image 1](#). These three scripts are very similar. However, they use three different *relational operators* to compare the values of the left and right operands and they set the values in three different output variables accordingly.

I will walk you through the construction of one of the scripts with a detailed explanation of each step in the process. You should be able to extend that explanation to the other two scripts.

An if-else block

I'm going to walk you through the process of creating one of the bottom three scripts shown in [Image 2](#).

[Image 3](#) shows the result of

- Selecting the **button** sprite in the lower-right panel of [Image 1](#).
- Selecting **Control** in the upper-left panel.
- Dragging the block labeled **when button clicked** into the center panel.

- Dragging an **if-else** block from the lower-left panel into the center panel.
- Connecting the **if-else** block to the block that is labeled **when button clicked** .

*By the way, in case I forgot to tell you before, you can set the name of the sprite to **button** by clicking in the dark gray area in the upper-right of [Image 3](#) and typing the name of the sprite.*

Image 3. Starting to use an if-else block.

Figure

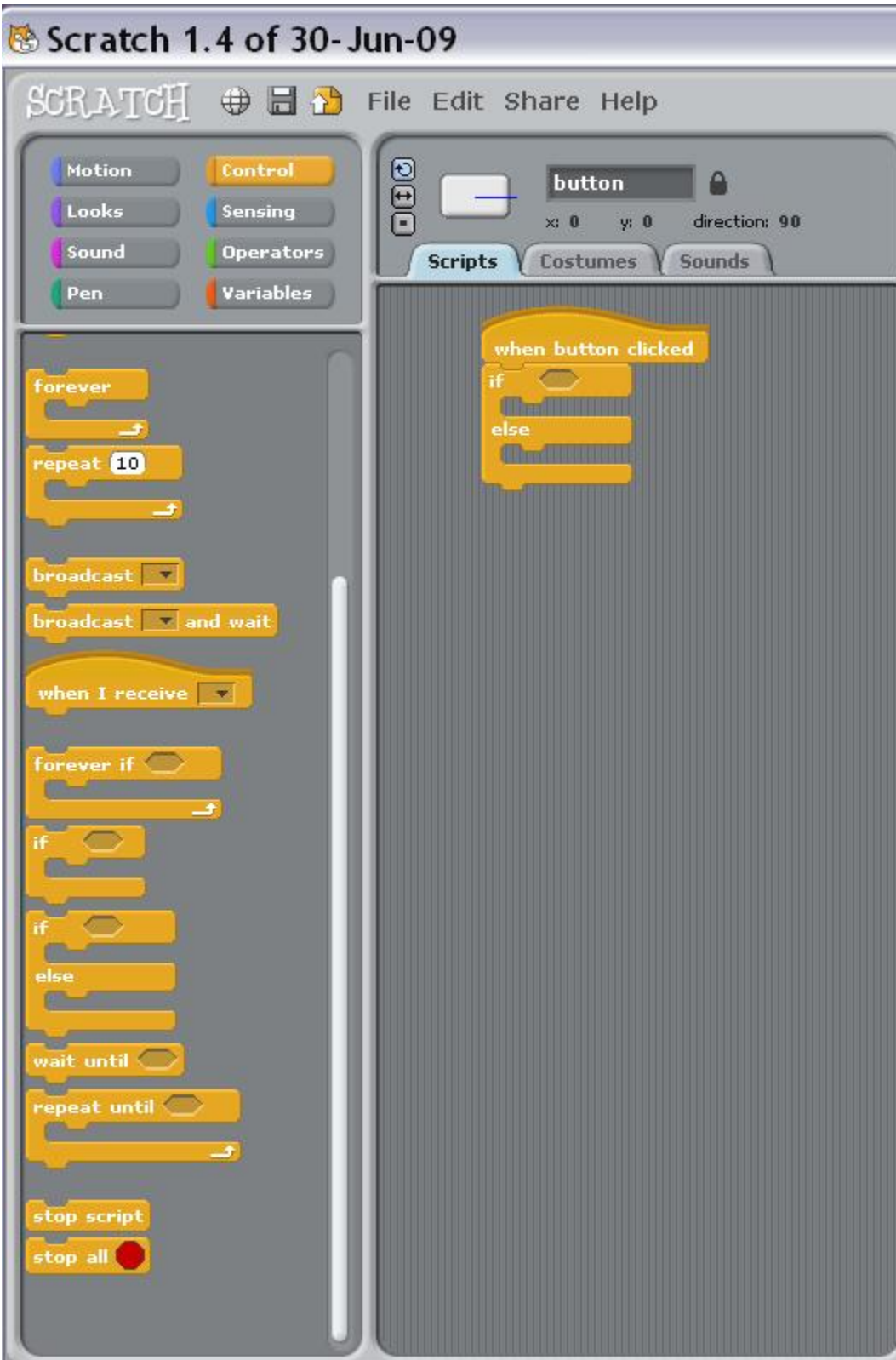


Image 3. Starting to use an if-else block.

Need to put a conditional clause in the pocket

Note the depressed pocket immediately to the right of the word **if** on the **if-else** block. We must drop another block of the correct shape into this pocket. The code in that block must evaluate to either true or false.

If that code evaluates to true, the code that we will insert into the mouth of the block immediately below the word **if** will be executed.

If the code that we drop into the pocket evaluates to false, the code that we insert into the mouth of the block immediately below the word **else** will be executed.

Where are the blocks with the correct shape?

There are two buttons (*possibly more*) that we can click in the upper left in [Image 3](#) that will expose blocks with shapes matching the pocket immediately to the right of the word **if** in [Image 3](#):

- Sensing
- Operators

We used a block having the correct shape from the **Sensing** group in the earlier module titled [Scr0130: Sequence, Selection, and Loop](#). An image of the blocks exposed by the **Sensing** button is shown in [Image 12](#) in that earlier module.

Will use blocks from the Operators group

In this module, we will use three of the blocks that are exposed by clicking the green **Operators** button as shown in [Image 4](#).

Image 4. Green blocks exposed by clicking the Operators button.

Figure

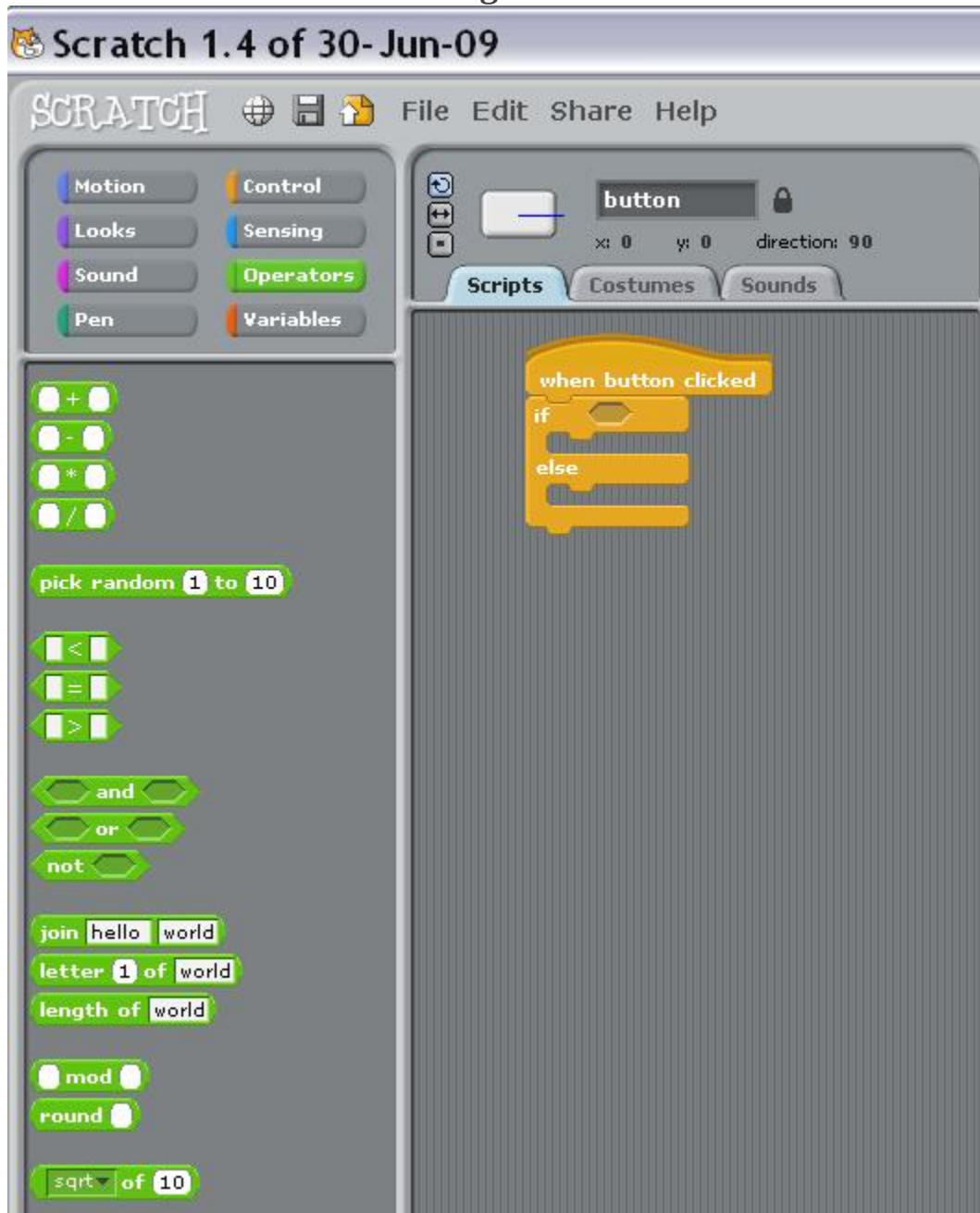


Image 4. Green blocks exposed by clicking the Operators button.

Six blocks have the correct shape

As you can see in [Image 4](#), there are six blocks of the correct shape exposed by clicking the **Operators** button. Those blocks are identified as follows:

Note:

```
< (less than)
= (equal to)
> (greater than)
and
or
not
```

We will refer to the first three blocks above as ***relational operators*** . (*They are used to evaluate the relationship between two values.*) We will refer to the other three blocks in the above list as ***logical operators*** , and will deal with them in a future module.

Relational operators in other languages

Many other modern programming languages also include the following three relational operators:

Note:

```
>= (greater than or equal)
<= (less than or equal)
```



```
!= (not equal)
```

Although these operators are a great convenience, they are not essential. We can get by with only the three that are provided by Scratch.

Also, in many other programming languages, a pair of equal characters (==) is used for the *equal to* operator instead of the single equal character (=) used in Scratch.

Three steps at once

[Image 5](#) shows the result of having taken three more steps in the development of the program:

1. I dragged the green **less than** (*indicated by a left angle bracket*) block and dropped it into the pocket immediately to the right of the word **if** .
2. I dragged the orange block labeled **set...to** for the variable named **LessThan** and inserted it immediately below the word **if** . I also set its literal value to 1.
3. I dragged another copy of the orange block labeled **set...to** for the variable named **LessThan** and inserted it immediately below the word **else** . I left its literal value as the default value of zero.

Image 5. Intermediate stage in construction of if-else script.

Figure

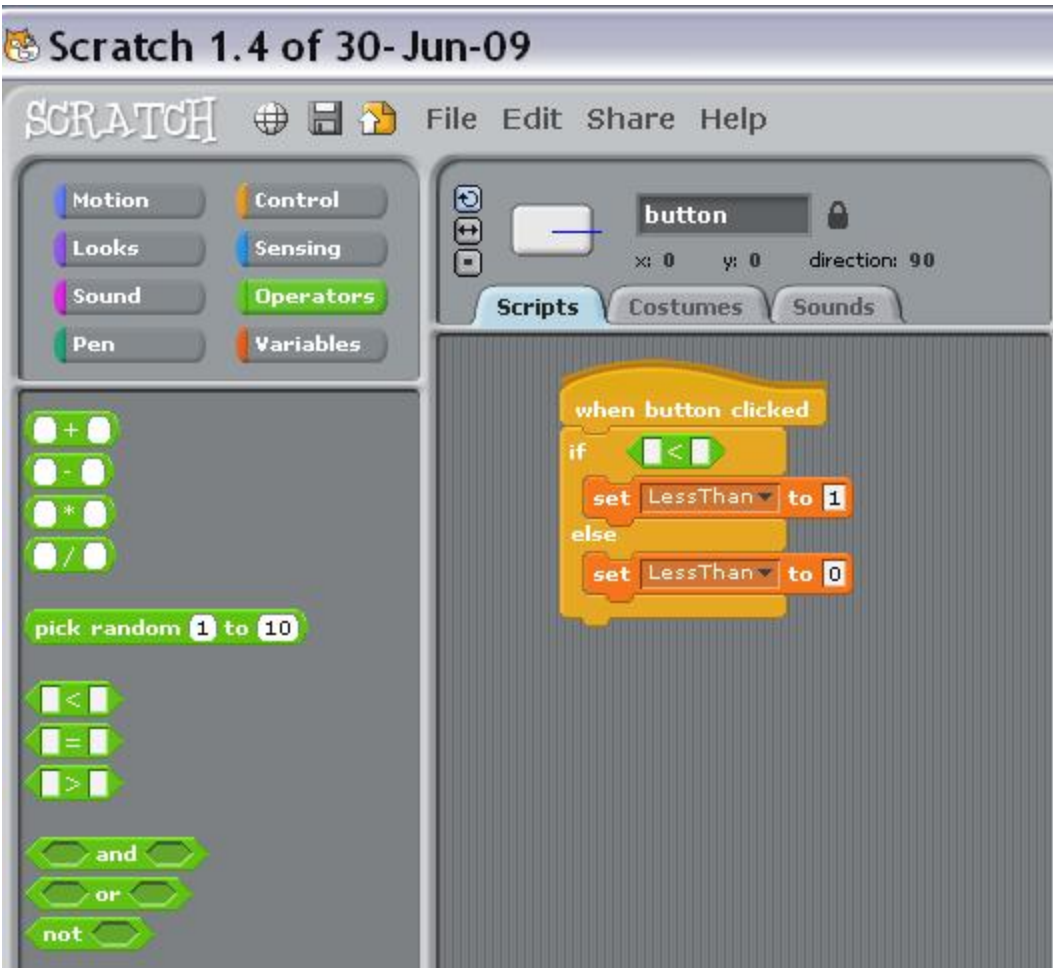


Image 5. Intermediate stage in construction of if-else script.

Still need to complete the conditional clause

The script in [Image 5](#) includes the action to be taken if the relational expression following the word **if** evaluates to true. It also includes the action to be taken if the relational expression evaluates to false. (*This relational expression is often referred to as a conditional clause.*) .

However, we still haven't completed the conditional clause in [Image 5](#). We don't want to test for the relationship between the literal values blank and

blank as shown in [Image 5](#). Instead we want to test the relationship between the values of the variables named **LeftOperand** and **RightOperand** as shown in the corresponding script in [Image 2](#).

Complete the relational expression

The next and final step for constructing this script is to:

- Click the **Variables** button to expose the variables as shown in [Image 1](#).
- Drag the block for the variable named **LeftOperand** and drop it in the white box immediately to the left of the left angle bracket operator in [Image 5](#).
- Drag the block for the variable named **RightOperand** and drop it in the white box immediately to the right of the left angle bracket operator in [Image 5](#).

Once we do that, we will have finished the construction of the second script from the top in [Image 2](#).

Completing the program

Following that, we need to go through essentially the same process to construct the bottom two scripts in [Image 2](#), using the other two relational operators in [Image 4](#) along with the variables named **LeftOperand** and **RightOperand**.

A screen shot of the program output

[Image 6](#) shows a screen shot of the left portion of the Stage for a given set of values for the variables named **LeftOperand** and **RightOperand**.

Image 6. Screen shot of the output from the program named Relational01.

Figure



Image 6. Screen shot of the output from the program named Relational01.

Interpretation of the results

The 0 showing in the variable named **LessThan** tells us that for these values, the following expression evaluates to false:

LeftOperand less than RightOperand

In other words, the left operand (5) is not less than the right operand (-3).

The 0 showing in the variable named **Equals** tells us that for these values, the following expression evaluates to false:

LeftOperand = RightOperand

In other words, the left operand (5) is not equal to the right operand (-3).

Finally, the 1 showing in the variable named **GreaterThan** tells us that for these values, the following expression evaluates to true:

LeftOperand greater than RightOperand

In other words, the left operand (5) is greater than the right operand (-3).

If you move the sliders to change the values in the left and right operand variables and then click the button, you are likely to get different results.

An online version of this program is available

A copy of this program has been posted online for your review (see [Resources](#) for the URL) . If you don't find the program using that URL, search the Scratch site for the user named **dbal** .

Run the program

I encourage you to use the information provided above to write this program. Experiment with the code, making changes, and observing the results of your changes. Make certain that you can explain why your changes behave as they do.

Just for fun, see if you cause a short drum roll to be produced by your computer speakers each time the button is clicked.

I also encourage you to write the program described below.

Student programming project

Write a project that meets the following specifications:

File name: Relational02.sb

Description: This program illustrates the use of the following *relational operators* in addition to arithmetic operators:

Note:

```
< (less than)
= (equal to)
> (greater than)
```

This is an upgrade to the program named **Relational01.sb**

The program creates the following six variables and a button and displays them on the screen as shown in [Image 7](#):

- LeftOperand - a slider
- RightOperand - a slider
- Offset - a slider
- LessThan
- Equals
- GreaterThan

Image 7. Screen shot of the output from the program named Relational02.

Figure

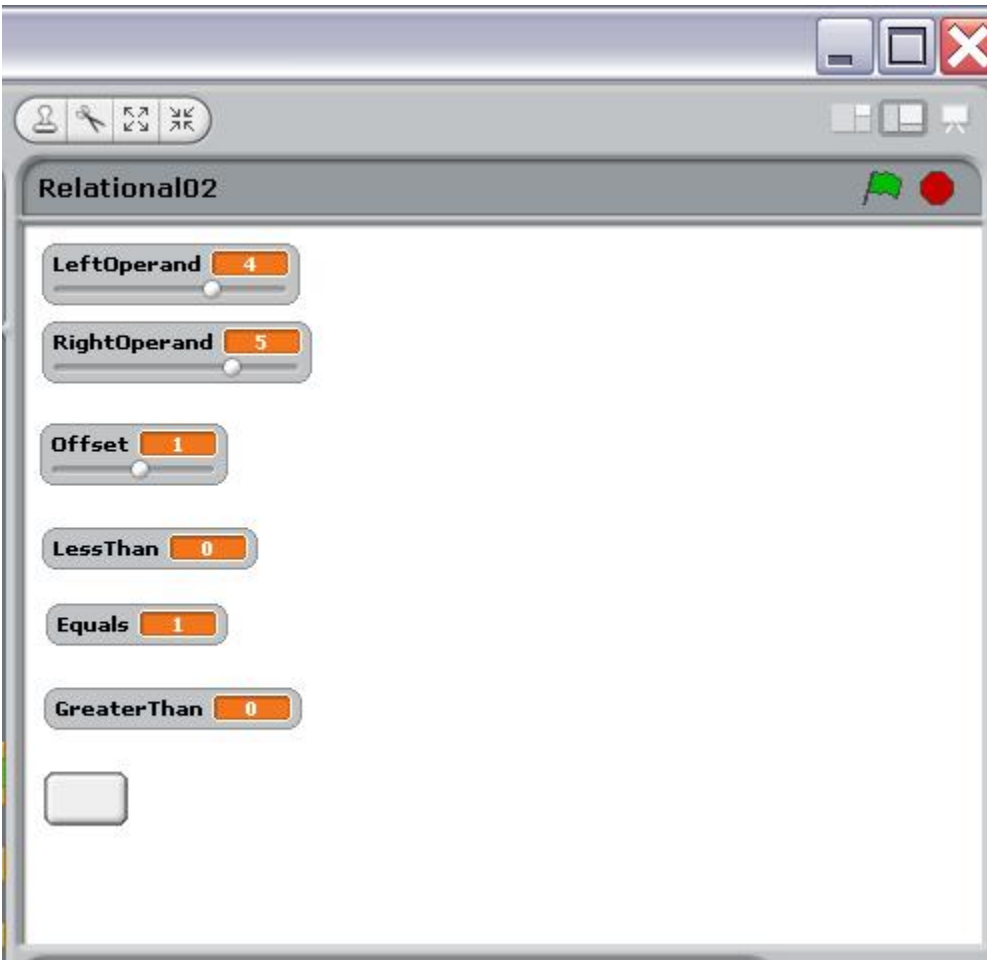


Image 7. Screen shot of the output from the program named Relational02.

Operation of the program

When the user clicks the green flag, the values of all six variables shown in [Image 7](#) are set to 0.

The user slides the three sliders to set the values of **LeftOperand** , **RightOperand** , and **Offset** .

When the user clicks the button, three separate event handlers on the button test the *sum of the left operand and the offset* against the right operand for *less than* , *equal to* , and *greater than* and display the results in the three variables having the corresponding names. A 0 is displayed if the value is false and a 1 is displayed if the value is true.

Make sure that your output matches the output shown in [Image 7](#) for the slider values shown in [Image 7](#).

A copy of this program has been posted online for your review (see [Resources](#) for the URL) . If you don't find the program using that URL, search the Scratch site for the user named dbal.

Summary

I began by providing a very brief review of material from previous modules.

Then I presented and explained the detailed steps required to write a Scratch program that uses the following relational operators:

Note:

```
< (less than)
= (equal to)
> (greater than)
```

Finally, I provided the specifications for a student-programming project for you to demonstrate your understanding of what you learned from the first program and from earlier modules.

Copies of both programs have been posted online for your review (see [Resources](#) for the URL) . If you don't find the program using that URL, search the Scratch site for the user named dbal.

What's next?

The previous module concentrated on *arithmetic* operators. This module explained the use of *relational* operators. The next module will explain the use of *logical* operators. Future modules using the Scratch programming language will continue to deal with *selection* and will also deal with *loops* .

Resources

General resources

- [Scratch home](#)
- [Scratch download page](#)
- [Scratch tutorial - Dance Tutorial](#)
- [Scratch forums](#)
- [Son of String Art](#)
- [Scratch explanatory video](#)
- [Scratch tutorials from MIT](#)
- [Scratch tutorial - Digital Art for All](#)
- [Scratch for Budding Computer Scientists](#)
- [Learn Scratch at learnscratch.org](#)
- [Scratch Tutorial - Space Shuttle Mission STS-2020](#)
- [SCRATCH TUTORIALS ICT In Primary Education 2012-2013](#)
- [Scratch Wiki](#)
- [Scratch WikiTable of ContentsWebsite - Scratch Wiki](#)
- [Scratch WikiTable of ContentsTutorials - Scratch Wiki](#)
- [Scratch WikiTable of ContentsProgram - Scratch Wiki](#)
- [Tutorial about Variables - Scratch Wiki](#)
- [Scratch support MIT web site](#)
- [Scratch resources at Scratch.ie](#)
- [ScratchEd Resources](#)
- [Scratch project Day Dream](#)

- [Scratch project Son of String Art](#)
- [Scratch Project Scratch Tutorial](#)

Programs used in this collection

- [Variable01](#) - Online version of program
- [Variable02](#) - Online version of student programming project
- [Variable03](#) - Online version of student programming project
- [IfSimple01](#) - Online version of program
- [IfWithVar01](#) - Online version of student programming project
- [Arithmetic01](#) - Online version of program
- [Arithmetic02](#) - Online version of student programming project
- [Relational01](#) - Online version of program
- [Relational02](#) - Online version of student programming project

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Scr0150: Relational Operators
- File: Scr0150.htm
- Published: 03/30/13

Note: Disclaimers:

Financial : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.

I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module. In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.

Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-